ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ - ΕΥΦΥΗ

ΣΥΣΤΗΜΑΤΑ ΚΑΙ IoT

ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΑΝΟΜΗΣ ΠΟΡΩΝ ΣΕ ΔΙΚΤΥΑ IOT ΒΑΣΙΣΜΕΝΑ ΣΕ ΤΕΧΝΟΛΟΓΙΕΣ ΥΠΟΛΟΓΙΣΤΙΚΗΣ ΟΜΙΧΛΗΣ (FOG COMPUTING)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΛΟΚΑ ΔΗΜΗΤΡΙΟΥ

ΑΜ: 8117010

Επιβλέπων: Δρ. Απόστολος Ξενάκης

ΛΑΡΙΣΑ 2020

ΠΕΡΙΛΗΨΗ

Στην ταχέως εξελισσόμενη εποχή της Πληροφορικής και των εφαρμογών της βιομηχανικής επανάστασης 4.0, υπάρχει συνεχής ανάγκη εκμετάλλευσης μεγάλου όγκου δεδομένων και πληροφορίας σε πραγματικό χρόνο.

Οι 3 πυλώνες τεχνολογιών δικτύωσης της βιομηχανίας 4.0, δηλαδή οι τεχνολογίες IoT, Edge και Fog Computing, έχουν φέρει επανάσταση στην ταχύτατη συλλογή, δυναμική επεξεργασία και λήψη αποφάσεων τόσο σε τοπικό (EDGE,FOG) όσο και σε διαδικτυακό επίπεδο (CLOUD).

Η συλλογή των δεδομένων, η οποία γίνεται μέσω ασύρματων δικτύων αισθητήρων (WSN), διαδραματίζει, λόγω της πολυμορφίας της φύσης των δεδομένων και της ποικιλίας των επιλεγμένων παραμέτρων, (ταχύτητα, αξιοπιστία μεταφοράς ,βέλτιστη αποθηκευτική ικανότητα ή γρήγορη επεξεργασία κλπ.) κρίσιμο ρόλο στις τεχνολογίες αυτές. Γι' αυτό είναι θεμιτή η κατάλληλη κατανομή πόρων μεταξύ των συσκευών Fog.

Σκοπός της παρούσας πτυχιακής εργασίας είναι η μελέτη, ανάλυση και σύγκριση αποδοτικών αλγορίθμων κατανομής πόρων σε δίκτυα IoT, βασισμένα σε τεχνολογίες υπολογιστικής ομίχλης (fog computing). Θα μελετηθεί σενάριο IoT για εφαρμογές βιομηχανίας 4.0, με δυνατότητες fog computing, και μέσω προσομοιώσεων βασισμένες σε κατάλληλα κριτήρια (π.χ. ενέργειας, τοπολογίας, τρόποι επικοινωνίας).

ABSTRACT

In the rapidly evolving era of Information Technologies and Industry 4.0 there is a constant need to exploit large amounts of data and information in real time.

Three of the main industry 4.0 pillars IoT, EDGE and Fog Computing technologies, have brought revolution in the procedures of rapid collection, dynamic processing and decision making both at local (EDGE, FOG) and online level (CLOUD).

WSN data collection has a critical role in these technologies due to the diversity of the nature of the data and the variety of selected parameters (speed, transfer reliability, optimum storage capacity or fast processing, etc.). In any case an appropriate allocation of resources is demanded.

The purpose of this paper is to study and analyze efficient resource allocation algorithms in IoT networks, with fog network computing. A fog computing IoT scenario will be analyzed and through simulation and appropriate criteria (energy, topology, way of communication between nodes) resource allocation algorithms will be studied. A bibliographic database will be compared, and optimum placement and collaboration scenarios will be proposed for Fog to IoT nodes and Fog to Cloud nodes. This simulation will be implemented with the iFogSim program.

ΕΥΧΑΡΙΣΤΗΡΙΑ

Για την παρούσα πτυχιακή εργασία θα ήθελα να ευχαριστήσω άτομα του συγγενικού και φιλικού κύκλου καθώς και αρκετούς από τους καθηγητές του μεταπτυχιακού IoT and Smart systems χωρίς την συμβολή των οποίων δεν θα ήταν δυνατή η ολοκλήρωσή της.

Πρωτίστως, θα ήθελα να ευχαριστήσω τον εισηγητή της πτυχιακής κ. Απόστολο Ξενάκη για την υπομονή και την βοήθεια που προσέφερε καθ' όλη την διάρκεια της εκπόνησής της.

Τους γονείς μου για την αδιάλειπτη ψυχολογική στήριξη και ηθική συμπαράσταση κατά τις μακροχρόνιες σπουδές μου σε προπτυχιακό και μεταπτυχιακό επίπεδο.

Τον συντονιστή υπεύθυνο καθηγητή του μεταπτυχιακού προγράμματος κ. Δημήτριο Βέντζα για την εμπιστοσύνη και συνεχή προσπάθειά του για την κάλυψη αναγκών στα πλαίσια του προγράμματος σπουδών.

Τέλος, όλους τους καθηγητές και βοηθούς – επιστημονικούς συνεργάτες χάρης την συμβολή των οποίων ήταν δυνατή η επίτευξη υψηλού επιστημονικού επιπέδου σπουδών.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

1.1 ΣΤΟΧΟΙ ΚΑΙ ΣΚΟΠΟΙ ΠΤΥΧΙΑΚΗΣ

Στη παρούσα διπλωματική εργασία βασικός στόχος είναι να εκτελεστεί πειραματική προσομοίωση ενός σεναρίου σε υπολογιστικό δίκτυο μιας εφαρμογής βιομηχανίας 4.0 και να εξαχθούν χρήσιμα συμπεράσματα από αυτή την προσομοίωση. Για την προσομοίωση αυτή θα γίνει χρήση αποδοτικών αλγορίθμων κατανομής πόρων σε εφαρμογές Fog Computing, σε δίκτυα IoT.

Αρχικά θα γίνει μια σύντομη θεωρητική αναφορά στη βιομηχανία 4.0 και ιδιαίτερα στους 3 πυλώνες τεχνολογιών δικτύωσης, τις τεχνολογίες IoT, EDGE και Fog Computing. Θα αναφερθούν οι ***βασικές ανάγκες*** με βάση τις οποίες προέκυψε η τεχνολογική εξέλιξη και τις ***καινοτομίες*** που επιφέρει κάθε μία από αυτές. Επίσης, θα αναφερθούν παραδείγματα πραγματικών εφαρμογών στο κόσμο (Real World Applications) της βιομηχανίας 4.0, ώστε να καταστούν κατανοητές οι νέες τεχνολογίες στην πράξη.

Στο Β' Κεφάλαιο θα γίνει αναφορά στην πολιτική κατανομής πόρων στα υπολογιστικά δίκτυα (resource allocation policy).Θα γίνει ανάλυση των αλγορίθμων κατανομής πόρων. Σκοπός είναι να επιλεχθούν αλγόριθμοι κατανομής πόρων με βέλτιστη λειτουργικότητα σε εφαρμογές Fog Computing. Θα γίνει επιλογή με βάση τις επιλεγμένες παραμέτρους (ταχύτητα, αξιοπιστία, αποθηκευτική ικανότητα ή γρήγορη επεξεργασία).Για τους αλγόριθμους που θα επιλεχθούν θα γίνει προσομοίωση με την εφαρμογή iFogSim.

Στο Γ' Κεφάλαιο θα αναλυθεί ο προσομοιωτής iFogSim και θα παραμετροποιηθεί με βάση τους επιλεγμένους αλγορίθμους κατανομής πόρων. Σκοπός είναι να εκτελέσουμε πείραμα προσομοίωσης σε πραγματικό σενάριο εφαρμογής βιομηχανίας 4.0.

Τέλος θα αναλυθούν μερικά σενάρια υλοποίησης και ένα πλαίσιο βελτιστοποίησης της προσομοίωσης με στόχο τα βέλτιστα τελικά αποτελέσματα.

1.2 ΟΡΟΛΟΓΙΑ IOT, EDGE , FOG COMPUTING

IoT – Internet of Things

Το Internet of Things ή Διαδίκτυο των Πραγμάτων είναι μια έννοια που αφορά τα αντικείμενα της καθημερινότητάς μας – από βιομηχανικές μηχανές μέχρι wearable συσκευές που χρησιμοποιούν ενσωματωμένους αισθητήρες για τη συλλογή δεδομένων και την ανάληψη κάποιας δράσης σε αυτά μέσα σε ένα δίκτυο.

Οι συσκευές αυτές όλες έχουν ως κοινό τη συλλογή και μετάδοση ροής δεδομένων (*streaming data*).

Τα *steaming data* είναι κατανεμημένα σε 2 βασικές κατηγορίες :

Α) Δεδομένα φυσικών μεγεθών (θερμοκρασία, πίεση, τάση ισχύος κλπ.) και

Β) Βιομετρικά Δεδομένα (παλμοί καρδίας, αρτηριακή πίεση, ποσοστό οξυγόνου κλπ.

Ο όρος διαδίκτυο των πραγμάτων επινοήθηκε στα τέλη της δεκαετίας του 1990 από τον επιχειρηματία Kevin Ashton και ιδρυτή του Auto-ID Center στο MIT, ήταν μέρος μιας ομάδας που ανακάλυψε τον τρόπο να συνδέσει τα αντικείμενα με το διαδίκτυο μέσω μιας ετικέτας RFID. Βέβαια στις μέρες μας η εξέλιξη ήταν εκθετική.

Η καινοτομία της τεχνολογίας IoT σε σχέση με τις μεθόδους ανάλυσης δεδομένων του παρελθόντος είναι ότι πλέον δε χρειάζεται ενδιάμεση αποθήκευση σε μέσο και μετέπειτα επεξεργασία των δεδομένων άλλα όλα πραγματοποιούνται σε πραγματικό χρόνο δυναμικά.

Στην περίπτωση των δεδομένων συνεχούς ροής (streaming data) όπως αυτά του IoT, τα μοντέλα και οι αλγόριθμοι είναι αυτοί που αποθηκεύονται και τα δεδομένα περνούν μέσα από αυτά για ανάλυση. Αυτό το είδος της ανάλυσης καθιστά δυνατό τον εντοπισμό και την εξέταση μοτίβων σε πραγματικό χρόνο.

Οι συσκευές IoT έχουν εξελιχθεί τα τελευταία χρόνια ακόμα περισσότερο αφού εκτός από πηγές δεδομένων συνεχούς ροής έχουν αντιστοιχία με ενεργοποιητές, διακόπτες κλπ. όπως θα δούμε σε χρήση στη βιομηχανία 4.0

Έτσι έχουμε καταλήξει στο γενικότερο όρο Smart Devices η Smart Networks με κύριες κατηγορίες των οποίων δομικοί λίθοι είναι οι IoT συσκευές.

α) Έξυπνο Δίκτυο Ενέργειας (Smart Power Grid)

β) Έξυπνη Πόλη (Smart City)

γ) Έξυπνη Γεωργία (Smart Agriculture)

δ) Έξυπνα Συστήματα Υγείας (Smart Health Systems)

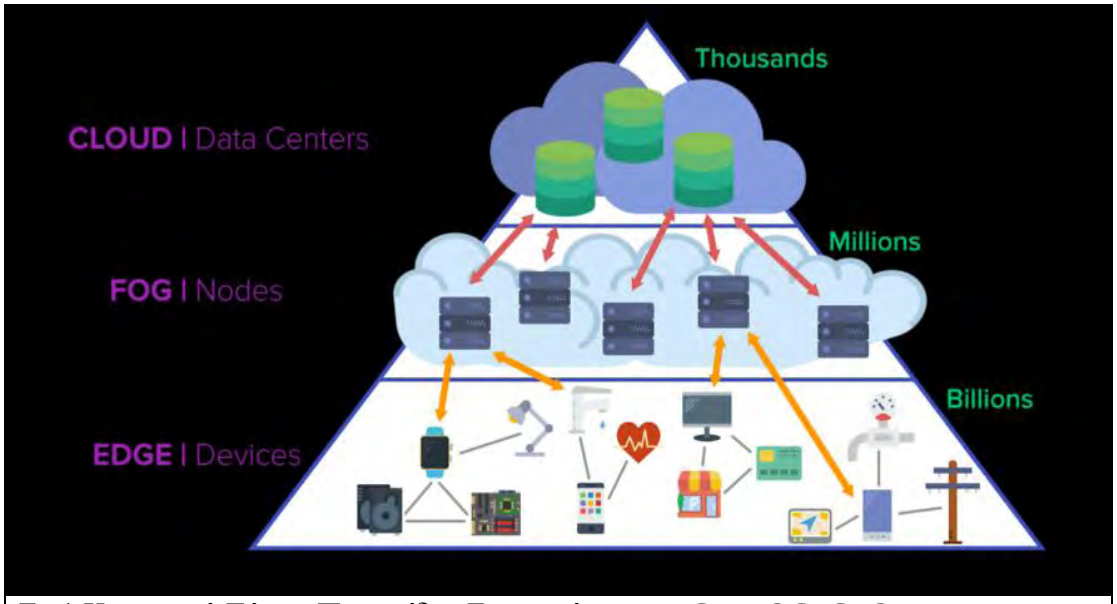
Edge Computing

Το *Edge Computing* είναι ένα κατανεμημένο μοντέλο υπολογιστών που επιτρέπει την επεξεργασία και την ανάλυση δεδομένων κατά μήκος ενός άκρου του δικτύου, πλησιέστερα στο σημείο της συλλογής τους, για να βελτιώσει τους χρόνους απόκρισης και να εξοικονομήσει bandwidth.

Με άλλα λόγια ένα υπολογιστικό δίκτυο το οποίο διαθέτει στο άκρο συλλογής του (σε περίπτωση μια βιομηχανίας τα assets της) κάποια δυνατότητα τοπικής αποθήκευσης και επεξεργαστική ισχύ για πρώιμη επεξεργασία δεδομένων.

Κύριος σκοπός των EDGE devices σε ένα δίκτυο είναι να ελαττωθεί ο όγκος των δεδομένων που προωθούνται από τα IoT devices προς το FOG ή CLOUD αφού μετά από μια πρώιμη επεξεργασία στο σημείο συλλογής (asset) μειώνεται ο αριθμός των μη χρήσιμων πληροφοριών και δεδομένων σε τεράστιο βαθμό.

Βασική ανάγκη που οδήγησε στην εξέλιξη του EDGE computing ήταν η μεγιστοποίηση ταχύτητας κυρίως στη μετάδοση δεδομένων κρίσιμης πληροφορίας και κρίσιμων συναγερμών ειδικότερα για περιπτώσεις IoT αισθητήρων που αφορούν την δημόσια ασφάλεια και την ανθρώπινη υγεία.



Σχ.1 Κατανομή Τύπου Πυραμίδας Συσκευών σε EDGE_FOG_CLOUD

Fog Computing

Με τον όρο *Fog computing* εννοούμε το σύνολο των υπολογιστικών δυνατοτήτων μεταφοράς αποθήκευσης και επεξεργασίας δεδομένων BIG DATA σε πραγματικό χρόνο σε (μικρότερο από διαδικτυακό) «τοπικό» επίπεδο (CISCO).[9]

Το Fog computing είναι μια σημαντική εξέλιξη, μεταξύ άλλων, της Διασύνδεσης υπολογιστικών συσκευών και αισθητήρων. Έχει εφαρμογές στα εμπορικά δίκτυα κινητής τηλεφωνίας, στις εμπορικές εφαρμογές λογισμικού αλλά ιδιαίτερα στην Βιομηχανία IoT ή IIoT. Εφαρμογή σε τομείς όπως η έξυπνη κατασκευή, το έξυπνο κτίριο, το έξυπνο δίκτυο, το πετρέλαιο και το φυσικό αέριο και γενικότερα στην Βιομηχανία 4.0

Το Fog Computing υλοποιείται στην πράξη με συσκευές Fog devices ή αλλιώς κόμβους (nodes). Κάθε κόμβος Node είναι ουσιαστικά ένα εξυπηρετητής ή σμήνος εξυπηρετητών υπολογιστών (server - server cluster) ο οποίος είναι συνδεδεμένος αμφίδρομα με τα Edge – IoT devices στο ένα άκρο και με το υπολογιστικό νέφος στο άλλο.(Σχ.1)

Βασική ανάγκη που οδήγησε στη εξέλιξη του Fog Computing είναι ότι τα cloud datacenters βρίσκονται σε απόσταση πολλαπλών βημάτων (hops) από τις πηγές δεδομένων IoT με αποτέλεσμα την καθυστέρηση στη διάδοση των δεδομένων.

Όπως περιγράφει παραστατικά ο όρος fog (ομίχλη) υπάρχει μία σημαντική αντιστοιχία με τον όρο cloud (νέφος). Η διαφορά του fog computing που συνήθως αποκαλείται και επέκταση του cloud computing έγγυται στο γεγονός ότι υπάρχει μεγαλύτερη πρόσβαση σε αποθήκευση, εφαρμογές και δεδομένα στους τελικούς χρήστες. Ωστόσο, το Fog Computing έχει πλησιέστερη γειτνίαση με τους τελικούς χρήστες και με μικρότερες γεωγραφικές κατανομές.

Τα κυριότερα πλεονεκτήματα του FOG Computing είναι:

1) Μικρή καθυστέρηση
2) Χαμηλό κόστος υλοποίησης
3) Ευκολία συντήρησης σε βάθος χρόνου
4) Εύκολη Επέκταση
5) Φορητότητα
6) Ευρεία υποστήριξη ετερογενών ασύρματων υποδικτύων.[8]

Πολύ συχνά οι έννοιες EDGE και FOG αποτελούν παρονομαστές λειτουργίας μέσα στα πλαίσια του ιδίου οικοσυστήματος δυναμικής αποθήκευσης και επεξεργασίας όπου η πληροφορία πηγάζει από τα IoT Devices και καταλήγει στο CLOUD. Δηλαδή το παράδειγμα λειτουργίας πυραμίδας του Σχ.1 δε είναι καθολικό

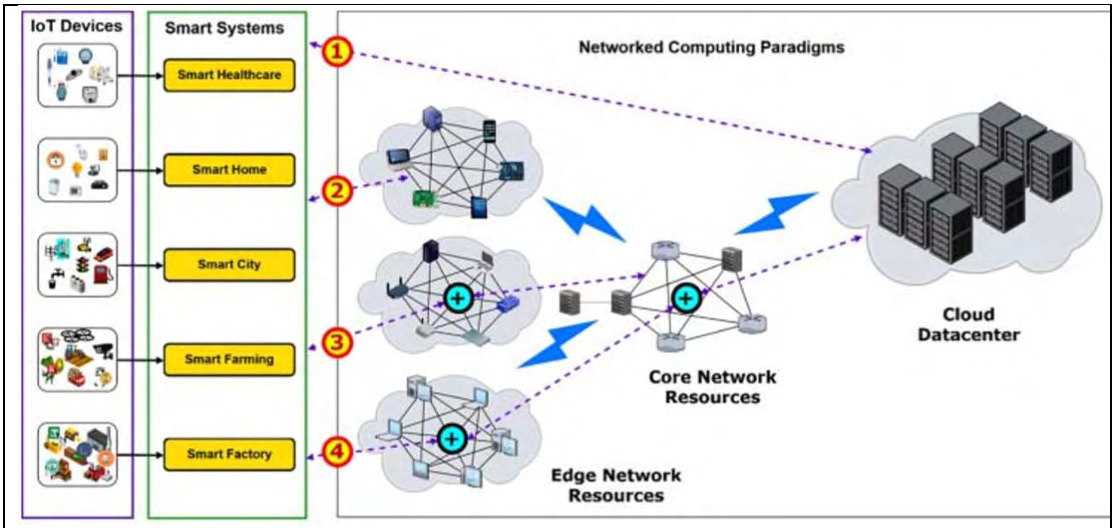
Σε ένα διάγραμμα λειτουργίας όπως φαίνεται στο παρακάτω σχήμα ενός έξυπνου δικτύου που περιλαμβάνει όλους τους τομείς

Δηλαδή:

α) Έξυπνα Συστήματα Υγείας, β) Έξυπνο Σπίτι, γ)Έξυπνη Πόλη δ) Έξυπνη Γεωργία ε) Βιομηχανία 4.0

Στους κόμβους που αντιστοιχούν σε κρίσιμες πληροφορίες και αποφάσεις που αφορούν δημόσιο κίνδυνο, συναγερμούς έκτακτης ανάγκης και ανθρώπινη υγεία έχουμε απευθείας επικοινωνία με το Cloud εφόσον δε έχουμε περιορισμό εύρους(bandwidth) (***περίπτωση 1***) ή μέσω EDGE Nodes (***περίπτωση 2***).

Ενώ στις περιπτώσεις 3,4 έχουμε μόνο FOG Nodes. [2]

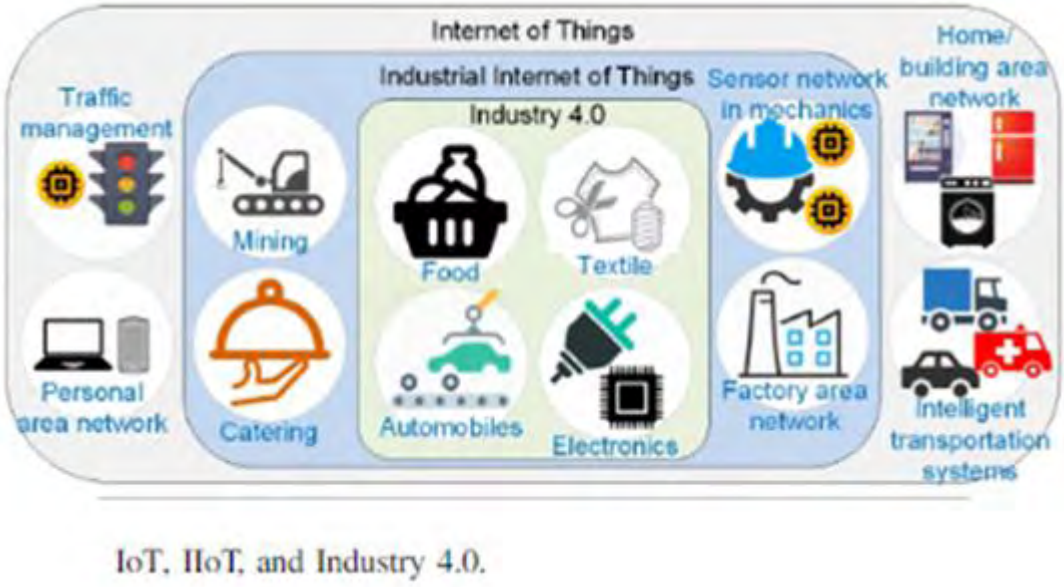


Σχ.2 Διαγραμμα σύνθετης συσχέτισης IoT-EDGE-FOG Nodes σε περιβάλλον CLOUD

1.3 INDUSTRY 4.0 ΠΑΡΑΔΕΙΓΜΑΤΑ ΚΑΙ ΕΦΑΡΜΟΓΕΣ

Με τον όρο Industry 4.0 (βιομηχανία 4.0) εννοούμε όλο το πλήθος των τεχνολογιών (IoT, Big Data, Cyberphysical Systems, WsN**) που έχουν προτυποποιηθεί με σκοπό την αυτοματοποίηση και ψηφιοποίηση της παραγωγής σε ένα σύγχρονο βιομηχανικό περιβάλλον.**

Οι βασικές ανάγκες που οδήγησαν στην τεχνολογική επανάσταση της βιομηχανίας 4.0 ήταν ή βελτιστοποίηση της ταχύτητας και ποιότητας αλλά κυρίως η οργάνωση στην συντήρηση της βιομηχανίας.



Σχ.3 IoT,IIoT και Βιομηχανία 4.0

Με την εισαγωγή στην βιομηχανική παραγωγή των τεχνολογιών τις Industry 4.0 είναι πλέον πραγματικότητα η Predictive Maintenance. (προγνωστική συντήρηση – προληπτική συντήρηση)

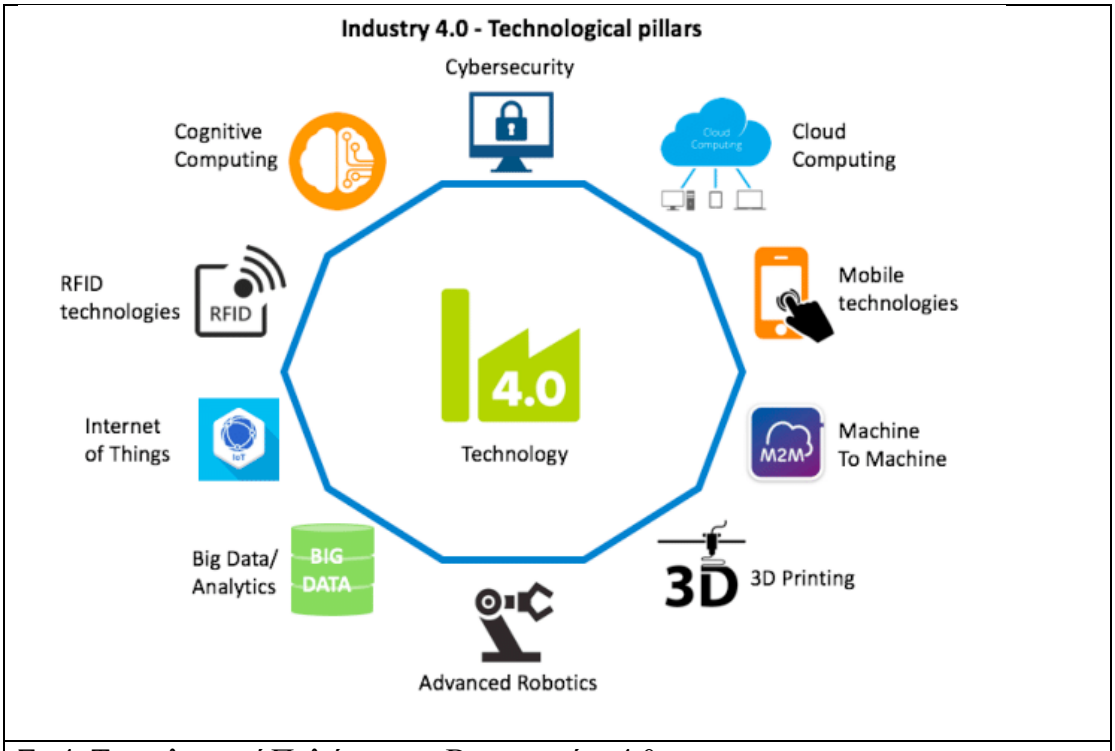
Πλέον, με τους σύγχρονους αισθητήρες υψηλής ακρίβειας του IIoT ,οι οποίοι συλλέγουν BIG DATA καθημερινά και τα προωθούν στο FOG αλλά και στο CLOUD για επεξεργασία και αποθήκευση, είναι δυνατόν να προβλέπονται οι αστοχίες υλικού με μεγαλύτερη ακρίβεια, γεγονός που θα μειώσει το μη προγραμματισμένο χρόνο

διακοπής λειτουργίας ενός μηχανήματος σε ένα σύστημα με σκοπό την συντήρηση του. Αυτή η μείωση του χρόνου διακοπής θα **αυξήσει τη συνολική αποδοτικότητα**, η οποία με τη σειρά της θα **βελτιώσει το οικονομικό όφελος της επιχείρησης**. Τα 2 τελευταία γεγονότα είναι μια σημαντική πρωτοπορία στην παραγωγή η οποία δε ήταν δυνατόν να πραγματοποιηθεί πριν εφαρμοστεί η προγνωστική συντήρηση.

Παράλληλα, αποτρέπονται καταστροφικές αστοχίες υλικού οι οποίες θα ήταν δυνατόν να καταστούν επιζήμιες είτε στην παραγωγή είτε ακόμα και στην υγεία του προσωπικού της βιομηχανικής εγκατάστασης.

Τεράστια ακόμα εξέλιξη είναι και το machine learning και τα κυβερνοφυσικά συστήματα.

Για να καταστούν δυνατά όλα αυτά απαραίτητες είναι σύγχρονες τεχνολογίες δικτύωσης του έμψυχου και άψυχου δυναμικού (assets) μια βιομηχανίας. **Οι τεχνολογίες δικτύωσης** χρησιμοποιούν IoT αισθητήρες και ενεργοποιητές (WSAN) σε WSNs (ασύρματα δίκτυα) για συλλογή όγκου μεγάλων δεδομένων(BIG DATA) και αποτελούν τους 3 βασικούς πυλώνες (technological pillars) της βιομηχανίας 4.0 τα IoT ,EDGE και Fog Computing.[5]



Σχ.4 Τεχνολογικοί Πυλώνες της Βιομηχανίας 4.0

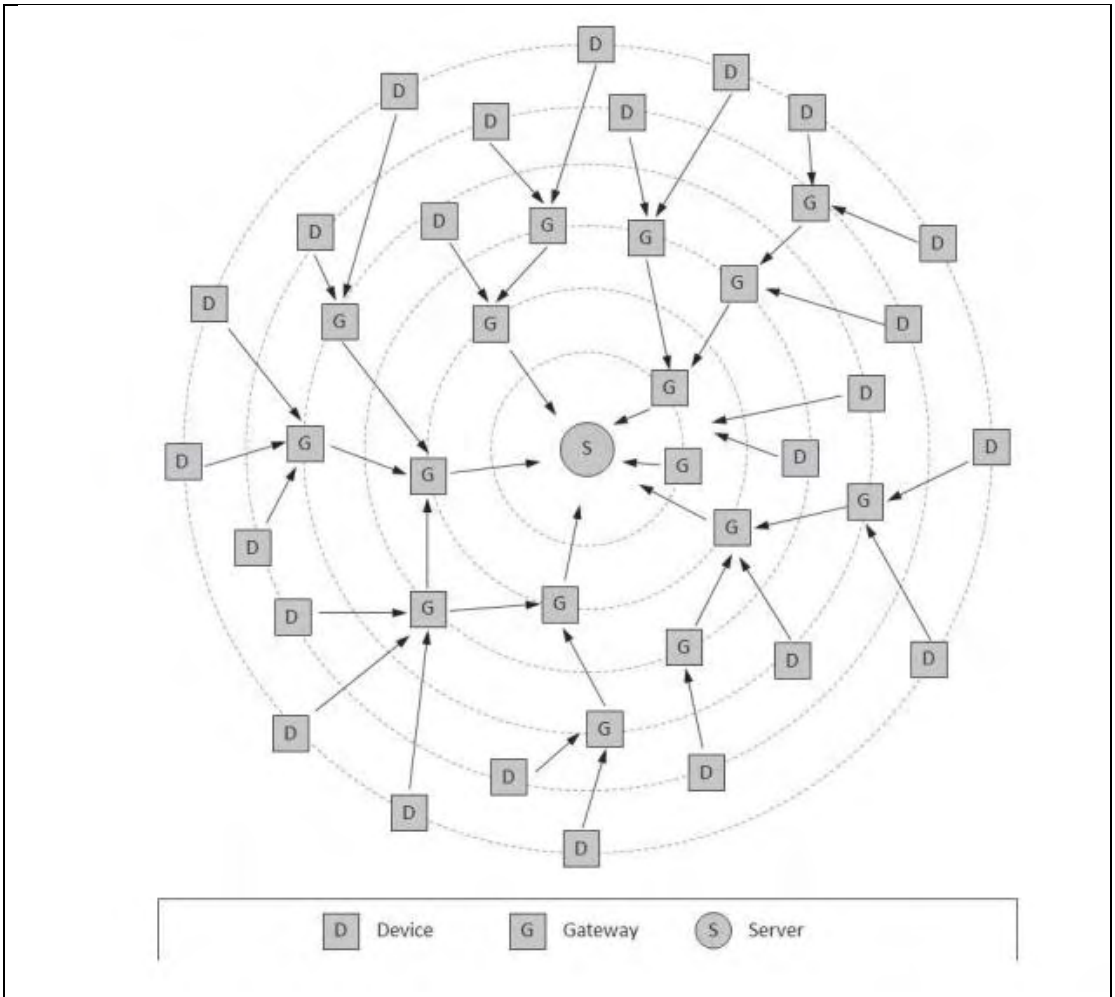
Ένα παράδειγμα εφαρμογής είναι η χρήση καμερών στο σύγχρονο περιβάλλον της βιομηχανίας 4.0. Μια κάμερα σύγχρονης τεχνολογίας υψηλής ευκρίνειας πλέον δε αποτελεί μόνο μια πηγή εικόνας όπως σε παρελθοντικές εποχές. Για την ακρίβεια μπορεί να αντικαταστήσει πλέον όλους του αισθητήρες μετά από training σε ένα neural network.Σε ένα σύγχρονο οικοσύστημα μιας βιομηχανίας 4.0 η πληροφορία σε μορφή streaming data είναι συνεχής και εκτελούνται άμεσα διεργασίες και εντολές βασισμένες στα πρότυπα και μοτίβα που έχουμε εμείς ορίσει.

IoT,EDGE και Fog Computing στην Industry 4.0

Η επεξεργασία δεδομένων στη βιομηχανία 4.0, απαιτεί τα περισσότερα καθήκοντα να εκτελούνται τοπικά λόγω απαιτήσεων **καθυστέρησης, ασφάλειας και δομημένων δεδομένων** που πρέπει να διαβιβάζονται μέσω του Διαδικτύου στις υπηρεσίες διαδικτύου και στο “νέφος» (cloud).

Για να επιτευχθεί αυτό χρειάζεται πρωτίστως τεράστια συλλογή δεδομένων από όσο των δυνατών περισσότερες πηγές πληροφορίας. Στην περίπτωση μια βιομηχανίας από όσο το δυνατών περισσότερα assets.Το ρόλο αυτό τον αναλαμβάνουν τα IoT devices.

Οι βιομηχανικές εφαρμογές βασίζονται συνήθως σε ένα *συγκεντρωτικό μοντέλο επεξεργασίας δεδομένων*. Δεδομένα που δημιουργούνται από βιομηχανικές συσκευές IoT(που μπορεί να διανέμονται γεωγραφικά σε μια ευρεία περιοχή) μεταφέρονται μέσω της υποδομής δικτύου σε κεντρικό χώρο αποθήκευσης. Το τελευταίο τυπικά αποτελεί το κέντρο δεδομένων (datacenter) όπου εκτελούνται εντατικές εργασίες επεξεργασίας δεδομένων.[6]



Σχ.5 Συγκεντρωτικό Μοντέλο Επεξεργασίας Δεδομένων σε Βιομηχανικές Εφαρμογές

Είναι εύκολο να παρατηρηθεί στη τοπολογία αυτού του συγκεντρωτικού μοντέλου(Σχ.5), ότι η επικοινωνιακή υποδομή του δικτύου μπορεί να γίνει εύκολα δυσμενής, καθώς ο αριθμός των συνδεδεμένων συσκευών IoT αυξάνεται.

Τα δεδομένα που ανταλλάσσονται θα καταναλώνουν πόρους επικοινωνίας, σε αναλογία με τον αριθμό των ζεύξεων (peers) και τη διάμετρο της τοπολογίας του δικτύου σύνδεσης.

Ιδιαίτεροι τύποι δεδομένων IoT (π.χ. ροές βίντεο, μετρήσεις σε πραγματικό χρόνο) επιβαρύνουν περαιτέρω την υποδομή του δικτύου σε σημείο να υπάρχει ακόμα και εξάντληση των πόρων.

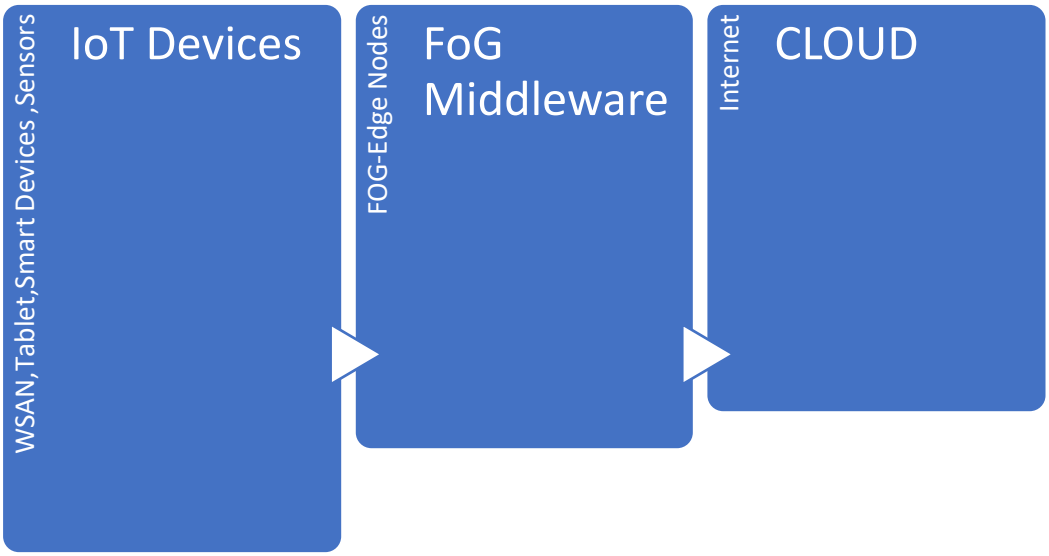
Επομένως, είναι ζωτικής σημασίας για αποτελεσματικούς μηχανισμούς για την ελαχιστοποίηση του φορτίου κίνησης δεδομένων που εγχέεται στην επικοινωνιακή υποδομή.

Άρα χρειάζεται πρώιμη επεξεργασία δεδομένων στο άκρα του δικτύου για να μειωθεί ο όγκος των δεδομένων και να βελτιωθεί η ταχύτητα. (EDGE Computing).

Στη συνέχεια, απαιτείται υποστήριξη middleware μεταξύ του βιομηχανικού περιβάλλοντος και των υπηρεσιών cloud / web. Σε αυτό το πλαίσιο, η **"ομίχλη"(fog)** είναι ένα **δυνητικό λογισμικό διαμεσολάβησης (middleware**)που μπορεί να είναι πολύ χρήσιμο για διαφορετικά βιομηχανικά σενάρια. Επιπλέον, καθώς τα μεγάλα βιομηχανικά δεδομένα είναι συχνά αδόμητα, μπορούν να καθαριστούν και να επεξεργαστούν στο fog τοπικά, προτού σταλθούν στο νέφος (cloud) .

Το υλικό μέρος του περιβάλλοντος FOG (συσκευές FOG - κόμβοι FOG) έχει διάρθρωση με ιεραρχική σειρά. Οι διατάξεις FOG χαμηλότερου επιπέδου συνδέονται απευθείας με συναφείς αισθητήρες και ενεργοποιητές. **Οι συσκευές FOG λειτουργούν σαν τα κέντρα δεδομένων** (datacenters) σε ένα πρότυπο υπολογιστικής «νέφους» και ως **παράγοντας** παροχή μνήμης, δικτύου και υπολογιστικών πόρων.

Κάθε FOG node εντάσσεται στο middleware περιβάλλον με συγκεκριμένες προδιαγραφές επεξεργαστικής ισχύος και κατανάλωσης ενέργειας (ισχύς σε κατάσταση λειτουργίας και σε αναμονή) που αντανακλά την ικανότητα και την ενεργειακή απόδοση του υλικού του μέρους.[6]



Σχ.6 Διάγραμμα Διάρθρωσης IoT-FoG-Cloud

Υλοποίηση του Fog Computing στο Industry 4.0

Ένα μεγάλο πλεονέκτημα του Fog Computing εκτός από την εγγύτητα στους τελικούς χρήστες και συσκευές είναι η υποστήριξη ετερογένειας που θεωρείται ως το πιο διακεκριμένο χαρακτηριστικό του IoT.[8]

Η υλοποίηση επομένως του Fog Computing στο βιομηχανικό τομέα είναι απαράμιλλα συνδεδεμένη με την ευρεία χρήση τεχνολογιών ασύρματης δικτύωσης αισθητήρων (WSN) τα τελευταία χρόνια.

Οι τεχνολογίες επικοινωνίας όπως το Zigbee, το χαμηλής ενέργειας Bluetooth (BLE), το πρωτόκολλο Internet Protocol Version 6 (IPv6) στο ασύρματο δίκτυο με τεχνολογικό πρότυπο αξιοπιστίας WiFi6 εφαρμόζονται διεξοδικά. Επίσης το ασύρματο δίκτυο 5G θα ωφελήσει σημαντικά τα επερχόμενα χρόνια με την τεράστια αύξηση ταχύτητας και κάλυψης δικτύου.

Τα ασύρματα δίκτυα χαμηλής κατανάλωσης (6LoWPAN) έχουν επίσης χρησιμοποιηθεί στο βιομηχανικό περιβάλλον. (Σχ.7)

Η ανάπτυξη και χρήση ασύρματων αισθητήρων (WSN) και ασύρματων δικτύων ενεργοποιητών / ηχείων αισθητήρων (WSAN) στις βιομηχανίες καθιστά δυνατή τη

βελτιστοποίηση της γραμμής παραγωγής με καλύτερη διαχείριση ποιότητας, ενεργειακή ασφάλεια, πρόβλεψη σφαλμάτων, σχεδιασμό προϊόντων και πιστότητα.

Παράλληλα στην εξέλιξη του Fog Computing συνέβαλε η τρομακτική αύξηση της επεξεργαστικής ισχύος των τοπικών εξυπηρετητών (servers) είτε λόγω σμίκρυνσης της λιθογραφίας είτε αύξηση ειδικά τα τελευταία χρόνια λόγω της παραλληλοποίησης της επεξεργασίας (multitasking) σε πολλούς υπολογιστές (Hive Computing) .

Με τις τεχνολογίες αυτές ήτο δυνατόν τα τελευταία χρόνια να υλοποιηθεί το fog computing ως middleware υπηρεσία όπως αναφέραμε και παραπάνω με τις παρακάτω προδιαγραφές :

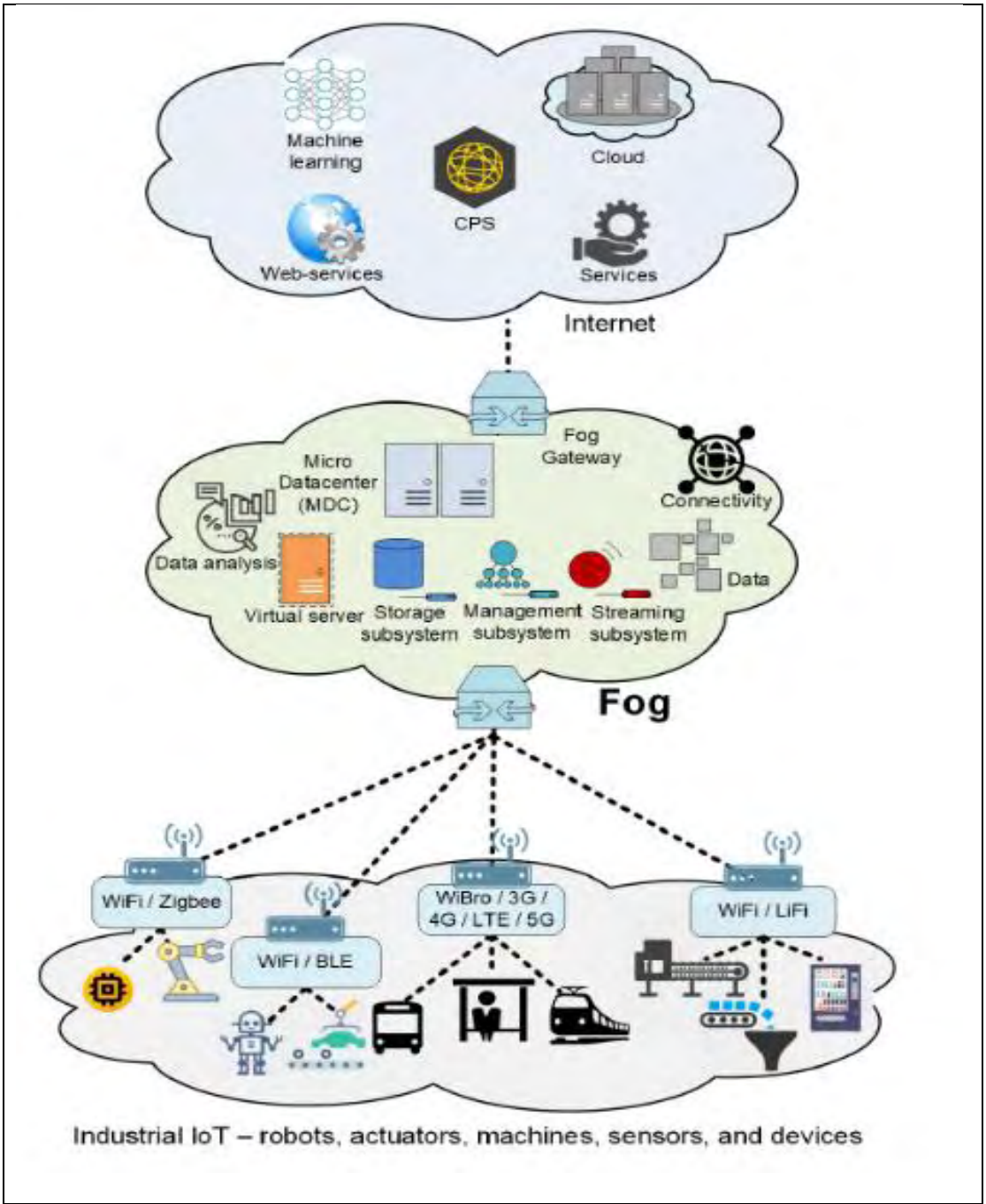
• *Επεξεργασία δεδομένων μεγάλου όγκου (**Big Data**) σε πραγματικό χρόνο για υψηλές επιδόσεις*

• *Συγχρονισμένη συλλογή δεδομένων από πολλούς τύπους αισθητήρων, ρομπότ και μηχανών*

• *Γρήγορη επεξεργασία των δεδομένων ανίχνευσης για την παραγωγή οδηγιών για τους ενεργοποιητές και τα ρομπότ μέσα στα προβλεπόμενα όρια καθυστέρησης*

• *Διασύνδεση ασύμβατων μεταξύ των αισθητήρων και μηχανών μέσω της μετάφρασης και της χαρτογράφησης του απαραίτητου πρωτοκόλλου*

• *Βέλτιστη διαχείριση ισχύος του συστήματος* [4]



Σχ.7 Network Technologies in IoT – Fog – Cloud Paradigm

IoT - Edge - Fog Computing και Πολιτική Κατανομής Πόρων

Με την ταχύτατη συλλογή και ενδιάμεση αποθήκευση – άμεση επεξεργασία σε τοπικό περιβάλλον που προσφέρει ο συνδυασμός των τεχνολογιών IoT,EDGE,FOG πλέον είναι γεγονός ένα έξυπνο περιβάλλον με δεδομένες αρχές - ρουτίνες λειτουργίας.

Όμως στον αντίποδα η συνύπαρξη όλων αυτών το διαφορετικών τεχνολογιών ασύρματης δικτύωσης στο ίδιο σύστημα με ταυτόχρονη απουσία ενός κοινού πρωτοκόλλου δικτύωσης δημιουργεί ιδιαίτερες προκλήσεις. [7]

Η διαχείριση των πόρων στο FOG Computing είναι πολύ περίπλοκη διαδικασία δεδομένου ότι εμπλέκει σημαντικό αριθμό κόμβων (FOG Nodes) διαφορετικών προτεραιοτήτων και περιορισμού πόρων. (diversial and resource constraint FOG nodes), με στόχο την κάλυψη της υπολογιστικής ζήτησης των IoT συστημάτων με κατανεμημένο τρόπο. Η ενσωμάτωσή τους στο υπολογιστικό «νέφος» (CLOUD) προκαλεί περαιτέρω διαφοροποιήσεις στη συνδυασμένη διαχείριση των πόρων. Η εκάστοτε συχνότητα ανίχνευσης των συσκευών IoT, η κατανεμημένη δομή εφαρμογής και ο συντονισμός τους επίσης επηρεάζουν τη διαχείριση των πόρων σε περιβάλλον υπολογιστικής ομίχλης [1]

Επεμβαίνοντας είτε «χειροκίνητα» είτε με την χρήση τεχνολογίας A.I στις παραμέτρους λειτουργίας του δικτύου αυτού μπορούμε να βελτιστοποιήσουμε την ταχύτητα παραγωγής, την μείωση της κατανάλωσης ενέργειας ακόμα και την προληπτική συντήρηση των μηχανημάτων.

Όπως προαναφέρθηκε τα μοντέλα και οι αλγόριθμοι είναι αυτοί που αποθηκεύονται και τα δεδομένα περνούν μέσα από αυτά για ανάλυση. Αυτό το είδος της ανάλυσης καθιστά δυνατό τον εντοπισμό και την εξέταση μοτίβων σε πραγματικό χρόνο. Επομένως, ανάλογα με τη πολιτική κατανομής πόρων η οποία θέλουμε να εφαρμοστεί (εξοικονόμηση ενέργειας, ταχύτητα κλπ.) προσαρμόζεται ο αλγόριθμος που έχει επιλεχθεί και αναλόγως επηρεάζεται ολόκληρο το οικοσύστημα της βιομηχανίας.

2. ΑΝΑΣΚΟΠΗΣΗ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΑΝΟΜΗΣ ΠΟΡΩΝ ΣΕ ΚΑΤΑΣΚΕΥΕΣ FOG

Η τεράστια ετερογένεια τελικών συσκευών (IoT) και τεχνολογιών ασύρματων υποδικτύων (wireless subnetwork technologies) σε εγγύτητα με τους τελικούς χρήστες αποτελούν μια πραγματικότητα σε ένα σύγχρονο περιβάλλον Βιομηχανίας 4.0

Από τις υπηρεσίες που προσφέρει το Internet of Things στο περιβάλλον αυτό σημαντικότερες είναι αυτές που κάνουν συλλογή δεδομένων με ταυτόχρονη λήψη αποφάσεων σε πραγματικό χρόνο, όπως πχ αναγνώριση προσώπου και γενική επιτήρηση της παραγωγής με κάμερες. Κύριο χαρακτηριστικό αυτών των υπηρεσιών είναι η δημιουργία μεγάλου όγκου δεδομένων (BIG DATA).

Για να υλοποιηθεί μεταφορά αυτού του τεράστιου σε όγκο δεδομένων σε πραγματικό χρόνο, στους απόμακρους CLOUD SERVER με τεράστια επεξεργαστική ισχύ όπου η ανάλυση δεδομένων είναι αποτελεσματική και άμεση, θα ήταν ιδιαίτερα δαπανηρό. Παράλληλα με το παραπάνω γεγονός , οι πληροφορίες μπορεί να απαιτούν προ-επεξεργασία σε χώρο μνήμης (π.χ. face recognition training). Ως εκ τούτου, η εφαρμογή της σε κόμβους IoT ακόμα και edge devices,που συνήθως είναι περιορισμένης ενέργειας με περιορισμένη επεξεργαστική ισχύ, μπορεί να προκαλέσει σημαντική υποβάθμιση της απόδοσης, δηλαδή χαμηλή ακρίβεια ή υψηλό ποσοστό σφάλματος.[10]

Επομένως, βλέπουμε ότι η χρήση όλων των στοιχείων IoT-Edge-Fog-Cloud είναι απαραίτητη για την λειτουργία του οικοσυστήματος και δε μπορεί κανένας παράγοντας ισχύος να παραλειφθεί. Επίσης είναι απαραίτητη η εύρυθμη λειτουργία μέσω σωστής κατανομής υπολογιστικών πόρων σε κάθε επίπεδο.

Σημαντικές επίσης είναι και οι προδιαγραφές λειτουργίας του οικοσυστήματος. Ενώ συνήθως δίνεται προτεραιότητα στην μικρή κατανάλωση ισχύος στις τελικές συσκευές και στη ταχύτητα λήψης των αποφάσεων δε είναι λίγες η φορές που έχουμε επιλογή κύριας η συμπληρωματική της κύριας παραμέτρου, διαφορετική από τις 2 παραπάνω.

Μερικές από τις βασικές παραμέτρους που ορίζουν την λειτουργία και την κατανομή πόρων σε κατασκευές FOG σε μια Βιομηχανία 4.0 μπορεί να είναι:

1. Ταχύτητα λήψης Αποφάσεων (π.χ. ενεργοποιητές) 2. Κατανάλωση Ενέργειας στις τελικές Συσκευές 3. Περιορισμός Εύρους Δεδομένων προς CLOUD (bandwidth restrictions) 4. Περιορισμός Χώρου αποθήκευσης στο FOG. 5. Quality of Services (QoS) για δεδομένα υψηλής εμπιστοσύνης ή κρίσιμων αποφάσεων.
Πιν.1 Βασικοί Παράμετροι Λειτουργίας Κατασκευών FOG

Κοινός παρονομαστής όλων των παραμέτρων αυτών είναι φυσικά ο **περιορισμός του κόστους** ως,

i) Λιγότερη Ενέργεια – Φθηνότερος Υλικοτεχνικός Εξοπλισμός

ii) Λιγότερος Χρόνος για την ίδια απόδοση

iii) Βελτίωση Υγείας -Συνθήκων Εργασίας-Ασφάλειας

Μερικές από τις θεωρίες προσέγγισης κατανομής πόρων με βάση τον πίνακα 1 θα παρουσιαστούν στη συνέχεια.

ΘΕΩΡΙΕΣ - ΑΛΓΟΡΙΘΜΟΙ

Εκτός του συγκεντρωτικού μοντέλου που αναλύθηκε σε προηγούμενο κεφάλαιο Σχ.5 έχουν αναπτυχθεί διάφορες θεωρίες προσέγγισης στην κατανομή των πόρων σε ένα υπολογιστικό δίκτυο τεχνολογίας FoG. Μερικές από αυτές είναι και οι παρακάτω:

1. PETRINETS
2. VIRTUALIZATION-VIRTUAL FOG
3. AOP- ADAPTIVE OPERATIONS PLATFORM

2.1 RESOURCE ALLOCATION STRATEGY WITH PRICED TIME PETRINETS

Μια κατασκευή υπολογιστικής ομίχλης μπορεί να αποσυντεθεί στη σύνθεση, εργασιών που εκτελούνται σε πόρους ομίχλης στους κόμβους FoG. Με βάση αυτό το συλλογισμό είναι δυνατόν όλες οι εργασίες εντός ενός δικτύου υπολογιστικής ομίχλης να αναχθούν σε δίκτυα Petri και σύνθεση-αλληλουχίες αυτών. [11]

Το Petri-Net (PN) είναι μια τριπλή ακολουθία (tuple) N=(P,T,F) όπου

P = {p1, p2, ..., pm}, m> 0, είναι ένα πεπερασμένο σύνολο θέσεων,

T = {t1, t2, ..., tn}, n> 0, είναι ένα πεπερασμένο σύνολο μεταβάσεων,

όπου $P \cup T = \varphi$ και $P \cap T = \varphi$,

$F \subseteq (P \times T) \cup (T \times P)$ είναι ένα σύνολο κατευθυνόμενων τόξων (σχέση ροής)

t ονομάζεται μετάβαση εισόδου του p και p ονομάζεται σημείο εξόδου του t εάν $(t, p) \in F$.

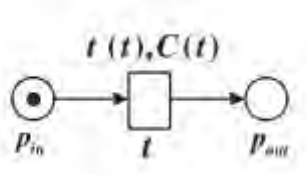
Στην τριπλή ακολουθία αυτή PN μπορεί να εισαχθεί ο παράγοντας του κόστους μέσω

α) Κόστος Χρόνου ολοκλήρωσης

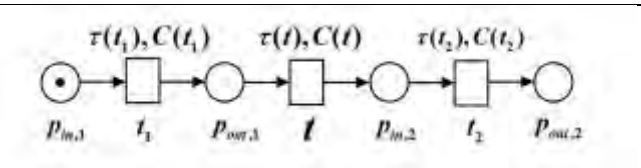
β) Κόστος τιμής.

Το παραγόμενο PN ονομάζεται PTPN.

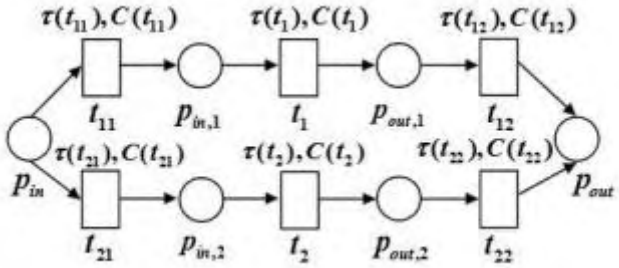
Μια αλληλουχία Priced PN (PTPN) εξυπηρετεί διάφορες εργασίες (tasks) όπως φαίνεται στο παρακάτω σχήμα.



PTPN 1 Εργασία



PTPN Συνεχόμενες Εργασίες



PTPN Κλάδος Εργασιών Σχ.8

Συγκεκριμένα για τη περίπτωση PTPN κλάδου για 2 tasks TS1 και TS2 που εφαρμόζεται σε συστοιχίες FoG είναι:

(Pi, Ti, Fi, M0i, τi,Ci)(i = 1, 2).

PTPN = (P, T, F, M0,τ,C) των εργασιών κλάδου TS1♦TS2 που ορίζονται ως:

1) P = P1∪P2∪{Pin, Pout}, T = T1∪T2∪{t11, t12, t21, t22},

F = F1 ∪ F2 ∪ {(pin, ti1), (ti1, pin,i), (pout,i, ti2),

(ti2, pout)}, (i = 1, 2), και M0 = M01 ⊕ M02 ⊕

M(pin, pout).

2) τ : T → R+, όπου τ (t1) = τ1(t1), τ (t2) = τ2(t2).

3) C : T → R+, όπου C(t1) = C1(t1),C(t2) = C2(t2).

Το Σχ. 8 δείχνει τη δομή PTPN δύο εργασιών κλάδου. Μπορεί να φανεί από το σχήμα ότι για δύο εργασίες κλάδου: εάν

εκτελείται στον πόρο t1, **το κόστος χρόνου** για την ολοκλήρωσή τους είναι

(t1) = τ (t11) + τ (t1) + τ (t12)

το κόστος τιμής για την ολοκλήρωσή τους είναι

ρ (t1) = C (t11) + C (t1) + C (t12)

Διαφορετικά, εάν εκτελούνται στον πόρο t2, το κόστος χρόνου για

ολοκλήρωση τους είναι

(t2) = τ (t21) + τ (t2) + τ (t22)

το κόστος τιμής είναι

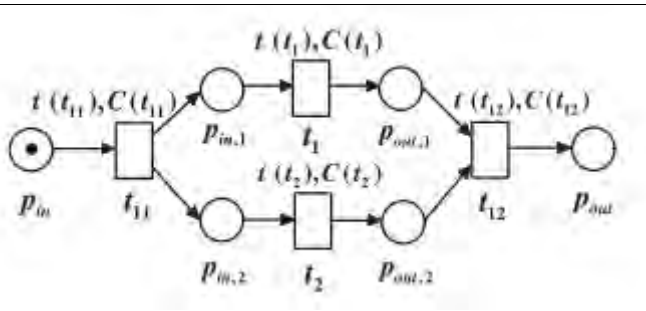
ρ (t2) = C (t21) + C (t2) + C (t22).

Δεδομένου ότι δεν μπορούμε να γνωρίζουμε εκ των προτέρων σε ποιον κλάδο τα καθήκοντα θα εκτελεστούν μπορούμε να εξετάσουμε το κόστος χρόνου και το κόστος τιμής ως

ω1 (t1) + ω2 (t2) και ω1ρ (t1) + ω2ρ (t2)

αντίστοιχα, όπου ω1 + ω2 = 1, ω1, ω2 είναι οι πιθανότητες να

εκτελούνται σε κάθε πόρο, αντίστοιχα.



Σχ.9 PTPN Συγχρονισμένες Εργασίες

1) P = P1 ∪ P2 ∪ {pin, pout}, T = T1 ∪ T2 ∪ {t11, t12},
   F = F1 ∪ F2 ∪ {(pin, t11), (t11, pin,i), (pout,i, t12), (t12, pout)}, (i = 1, 2),
   και M0 = M01 ⊕ M02 ⊕ M(pin, pout).
2) τ : T → R+, όπου τ (t1) = τ1(t1), τ (t2) = τ2(t2).
3) C : T → R+, όπου C(t1) = C1(t1),C(t2) = C2(t2).

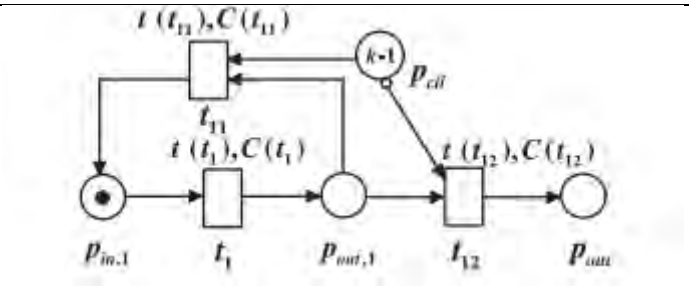
Το κόστος χρόνου για την ολοκλήρωσή τους είναι

τ (t11) + max {τ (t1), τ (t2)} + τ (t12)

Το κόστος τιμής για την ολοκλήρωσή τους είναι

C (t11) + C (t1) + C (t2) + C (t12).

Σε αυτή τη περίπτωση (Σχ.9), δεν υπάρχει εξάρτηση μεταξύ δύο εργασιών που εκτελούνται ταυτόχρονα, αλλά επιλέγεται ο μεγαλύτερος μεταξύ των 2 χρόνων ολοκλήρωσης των δύο εργασιών.



Σχ.10 PTPN βρόχου εργασιών k επαναλήψεων

Υποτίθεται ότι το PTPN της εργασίας TS1 είναι

(P1, T1,F1, M01, τ1,C1).

PTPN = (P, T, F, M0,τ,C) με εργασίες βρόχου ⊗TS1

1) P = P1 ∪ {pctl, pout}, T = T1 ∪ {t11, t12},
   F = F1∪ {(t11, pin,1), (pout,1, t11), (pout,1, t12), (pctl, t11), (pctl, t12), (t12, pout)}, και M0 = M01 ⊕ M(pctl, pout).
2) τ : T → R+, όπου τ (t1) = τ1(t1).
3) C : T → R+, όπου C(t1) = C1(t1).

το κόστος χρόνου για την ολοκλήρωσή τους είναι

kτ (t1) + (k - 1) τ (t11) + τ (t12)

το κόστος τιμής για την ολοκλήρωσή τους είναι

kC (t1) + (k - 1) C (t11) + C (t12).

Σενάριο Εφαρμογής

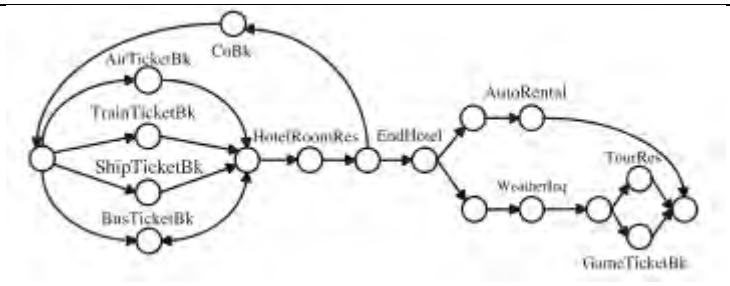
Ένα σενάριο εφαρμογής είναι δίκτυο FoG για τουριστικές επιχειρήσεις. Βασικές προϋποθέσεις

1) χαμηλός λανθάνοντας χρόνος,

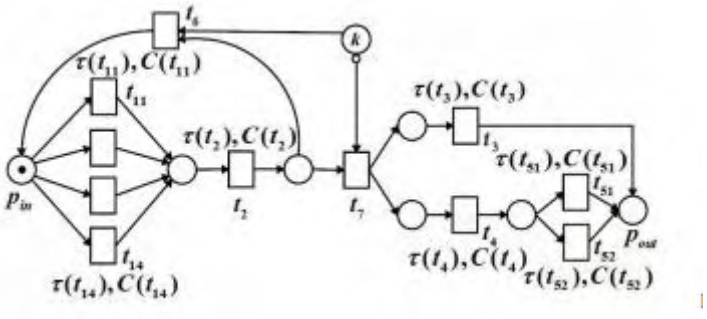
2) ευρεία γεωγραφική κατανομή

3) υψηλή κινητικότητα.

Το διάγραμμα ροής εργασιών (Σχ.11) και το μοντέλο εφαρμογής του δικτύου FoG που εφαρμόζεται ο αλγόριθμος PTPN(Σχ.12) παρουσιάζονται παρακάτω.



Σχ.11 Διάγραμμα Ροής Εργασιών



Σχ.12 Μοντέλο Εφαρμογής Αλγορίθμου PTPN

Παρατηρείται ότι η ροή εργασιών του δικτύου FoG μπορεί να αναλυθεί σε βρόχους PTPN κλάδου εργασιών, συγχρονισμένων εργασιών και εργασιών επαναλήψεων k.

Πολιτική Κατανομής Πόρων σε PTPN

Η πολιτική κατανομής πόρων περιστρέφεται γύρω από την διαμόρφωση των ροών Tuple. Ανάλογα με την μορφή του μοντέλου εφαρμογής αλγορίθμου PTPN που θα εφαρμοστεί καθορίζονται η κατανομή του κόστους και του χρόνου εκτέλεσης ενός κύκλου λειτουργίας του δικτύου και η ροή εργασιών.

Επομένως ο αλγόριθμος αυτός έχει εφαρμογή σε tuple και application based στρατηγικές όπου κυρίαρχο ρόλο διαδραματίζει η μορφή της αρθρωτής εφαρμογής ή εφαρμογών που θα εκτελεστούν στο υπολογιστικό δίκτυο της ομίχλης και οι μορφές των ροών των δεδομένων και όχι τόσο πολύ τα φυσικά χαρακτηριστικά ή η τοπολογία των FoG και EDGE Nodes του δικτύου.

2.2 VIRTUAL FOG: A VIRTUALIZATION ENABLED FOG FOR IOT

Σε ένα πολυεπίπεδο πλαίσιο υπολογιστικής ομίχλης για καλύτερη υποστήριξη εφαρμογών IoT, που περιλαμβάνει όλα τα επίπεδα κατά μήκος του Cloud-to-Things μια προσέγγιση εικονικοποίησης (Virtualization) παρουσιάζει εξαιρετικό ενδιαφέρον.

Ειδικότερα, η εικονικοποίηση αναφέρεται σε έναν μηχανισμό αφαίρεσης, στοχευμένο στην απόκρυψη λεπτομερειών της υλοποίησης και της κατάστασης ορισμένων υπολογιστικών πόρων από πελάτες των πόρων αυτών (π.χ. εφαρμογές, άλλα συστήματα, χρήστες κλπ.) Η εν λόγω αφαίρεση μπορεί είτε να αναγκάζει έναν πόρο να συμπεριφέρεται ως πλειάδα πόρων (π.χ. μία συσκευή αποθήκευσης σε διακομιστή τοπικού δικτύου), είτε πολλαπλούς πόρους να συμπεριφέρονται ως ένας (π.χ. συσκευές αποθήκευσης σε κατανεμημένα συστήματα).[8]

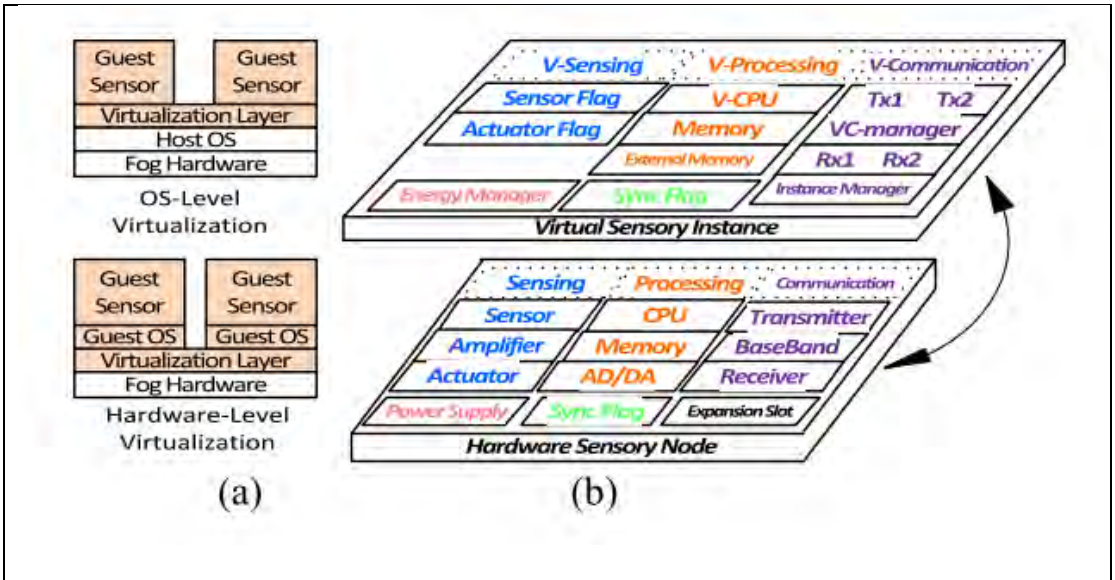
Στοιχεία Εικονικοποίησης Δικτύου IoT-FoG

A) Εικονικοποίηση Αισθητήρων

Ένα **εικονικό αντικείμενο (VO)** παρέχει μια σημασιολογική περιγραφή των δυνατοτήτων και ιδιοτήτων για το σχετικό αντικείμενο πραγματικού κόσμου. Παρά τις ετερογενείς λειτουργίες, οι φυσικοί αισθητήρες συνήθως ενσωματώνονται με έναν επεξεργαστή, μια μικρή ποσότητα μνήμης, μερικές φορές εξωτερική μνήμη τύπου flash, μονάδα τροφοδοσίας, μονάδα ανίχνευσης και ολοκληρωμένο κύκλωμα επικοινωνίας.

Όλα αυτά τα στοιχεία (μνήμη, επεξεργαστική ισχύς κ.λπ.) μπορούν ενδεχομένως να δημιουργηθούν και να υλοποιηθούν ως παρουσίες εικονικού λογισμικού (Virtual FoG instances), οι οποίες φιλοξενούνται σε κόμβους Fog με πραγματική φυσική υπόσταση.

Στο Σχ.13 παρουσιάζονται δύο κύριες ροές εικονικοποίησης αντικειμένων, δηλαδή, εικονικοποίηση σε επίπεδο λειτουργικού συστήματος και εικονικοποίηση σε επίπεδο υλικού.

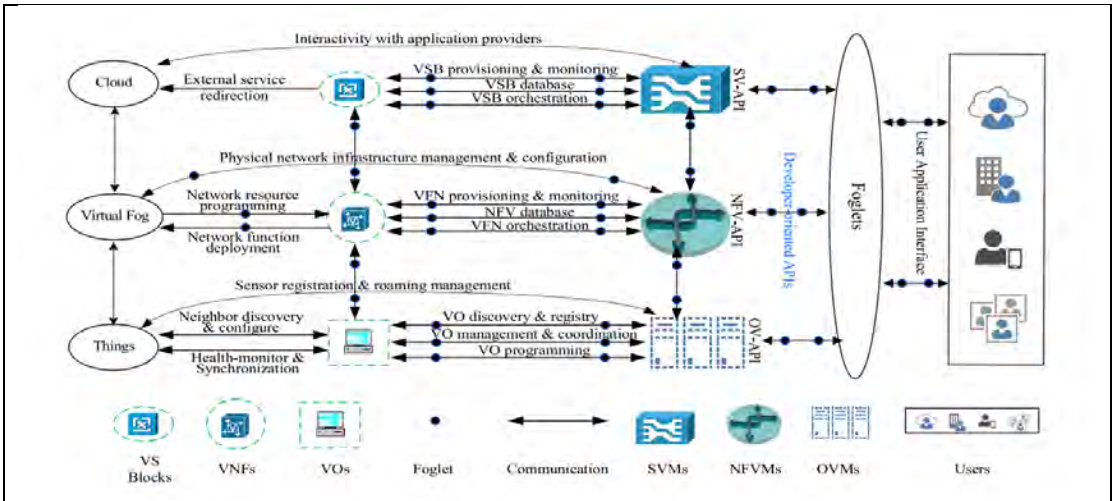


Σχ.13 a) Εικονικοποίηση Αντικειμένου σε Επίπεδο Λειτουργικού Συστήματος και σε Επίπεδο Υλικού

b) Εικονικοποίηση Αισθητήρα σε Fog Node

B) Διαχειριστής εικονικοποίησης αντικειμένων (OVM)

Ενώ ένα εικονικό αντικείμενο (VO) είναι ένα δομικό στοιχείο για τη βάση του επιπέδου εικονικοποίησης αντικειμένων, **ο διαχειριστής εικονικοποίησης αντικειμένων (OVM)** είναι υπεύθυνος για τη διαχείριση και ενορχήστρωση εικονικών και φυσικών οντοτήτων όπως απεικονίζονται στο Σχ.14



Σχ.14 Δίκτυο Υπολογιστικής ομίχλης με ενεργοποιημένη εικονικοποίηση

Αρχικά, ένας διαχειριστής Εικονικοποίησης Αντικειμένων εκτελεί διαχείριση κύκλου ζωής των αντίστοιχων VO, συμπεριλαμβανομένης της δημιουργίας και του τερματισμού τους. Επίσης είναι υπεύθυνος για την ανακάλυψη και την εγγραφή, την παρακολούθηση και συντονισμό, καθώς και ανάπτυξη και προγραμματισμό αυτών.

Παράλληλα με τα παραπάνω ένας OVM διατηρεί ένα αρχείο διαμόρφωσης των VO που είναι επεξεργάσιμο από τοπικούς και / ή απομακρυσμένους χρήστες. Μόλις καταχωρηθεί ένα εικονικό αντικείμενο σε έναν OVM, θα κατεβάσει το προφίλ διαμόρφωσής του, έτσι ώστε να εκτελείται άμεσα η αυτόματη διαμόρφωση και η αυτορρύθμιση για την ανάπτυξη υπηρεσιών. Για περαιτέρω επίτευξη επιχειρηματικής ευελιξίας, ένας OVM είναι εξοπλισμένος με διεπαφή προγραμματισμού εφαρμογών (**A**pplication **P**rogramming **I**nterface) λεγόμενο OV-API, μέσω του οποίου, οι χρήστες του δικτύου ομίχλης μπορούν να αλληλοεπιδράσουν γρήγορα με εικονικά αντικείμενα για διαχείριση και διαμόρφωση των τιμών των φυσικών αισθητήρων.

Διαχείριση Πόρων σε δίκτυο IoT-FoG με υπηρεσίες Εικονικοποίησης

Η αποτελεσματική διαχείριση πόρων είναι υψίστης σημασίας για την επίτευξη ανώτερης απόδοσης σε ένα περίπλοκο IoT οικοσύστημα. Για το σκοπό αυτό, ο OVM επιβλέπει την κατανομή πόρων όλων των VO στον κόμβο Fog, διασφαλίζοντας ότι κάθε VO έχει αρκετούς πόρους για να εξασφαλίσει την ποιότητα των υπηρεσιών (QoS). Εάν ο υποδοχέας (συσκευή ή πλήθος συσκευών FoG) ενός VO είναι υπερφορτωμένο, το OVM θα μετ' εγκαταστήσει το VO σε ένα νέο υποδοχέα με περισσότερους πόρους. Μπορεί επίσης να μειώσει την δέσμευση πόρων σε υποδοχείς VO που έχουν λιγότερες εργασίες να επιτελέσουν.

Με την αποτελεσματική αυτή διαχείριση είναι δυνατόν να εκτελεστούν πολύπλοκες διεργασίες και αρχιτεκτονικές δικτύου με δυναμικές προβλέψεις αλλαγής μεγέθους ή εφαρμογών-μοντέλου εκτέλεσης. Γεγονότα όπως μεγάλη ετερογένεια αισθητήρων, πολλαπλά εικονικά αντικείμενα σε ίδιους υποδοχείς FoG, καθώς και κίνδυνοι από καταστροφή ή δυσλειτουργία φυσικών αντικειμένων που αντιστοιχούν σε VO ,δεν αποτελούν πρόβλημα για την διαχείριση πόρων με Εικονικό Διαχειριστή Αντικειμένων (OVM)

Εάν οι συνολικοί φυσικοί πόροι είναι περιορισμένοι στον κόμβο Fog, μερικοί αλγόριθμοι όπως η κατανομή πόρων βάσει προτεραιότητας με συμφωνία επιπέδου υπηρεσίας (SLA) θα μπορούσαν να εφαρμοστούν. Για κάθε VO, εφαρμόζεται ένας μηχανισμός ουράς (queuing mechanism) για τις εισερχόμενες εργασίες. Ορισμένες τεχνικές ευρετικού προγραμματισμού (π.χ., first-come first-serve, round-robin κ.λπ.) εφαρμόζονται επίσης για εκτέλεση των εργασιών με τους πόρους που διατίθενται στον υποδοχέα FoG.

Επομένως στο μοντέλο αυτό παρατηρείται ότι υπάρχει κύρια εστίαση πάνω στα τεχνικά χαρακτηριστικά και την συνδεσμολογία μεταξύ των φυσικών οντοτήτων, καθώς ο OVM καθορίζει τις προτεραιότητες σε πόρους με βάση την υπηρεσία ποιότητας πάντοτέ με γνώμονα αυτά τα τεχνικά χαρακτηριστικά των φυσικών οντοτήτων που φιλοξενούν τα VO.

2.3 AOP- ADAPTIVE OPERATIONS PLATFORM

Η πλατφόρμα προσαρμοστικών λειτουργιών (AOP), έχει ως σκοπό την βελτίωση της αποτελεσματικότητας της εφαρμογής σε ένα δίκτυο υπολογιστικής ομίχλης λαμβάνοντας υπόψη τα μοντέλα αστοχίας εξοπλισμού για την ανάπτυξη και διαμόρφωση κάθε συγκεκριμένης υποδομής.[6]

Η πλατφόρμα διασυνδέεται με κατασκευαστές εξοπλισμού και τις επιχειρησιακές διοικήσεις υποδομών εποπτικού ελέγχου και απόκτησης δεδομένων (SCADA), προσφέροντας τις ακόλουθες δυνατότητες:

Α) Δυνατότητα στον κατασκευαστή του εξοπλισμού να:

• Διαχειριστεί τη δημοσίευση της πλατφόρμας του μοντέλου αποτυχίας που σχετίζεται με κάθε συγκεκριμένο τύπο εξοπλισμού.

• Διαχειριστεί τα δικαιώματα πρόσβασης κάθε δημοσιευμένου μοντέλου αποτυχίας.

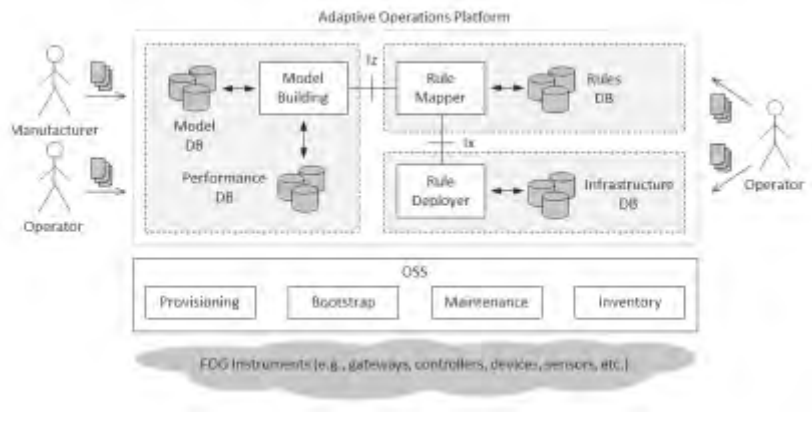
• Καταγράψει την πρόσβαση και τη χρήση κάθε δημοσιευμένου μοντέλου αποτυχίας.

Β) Δυνατότητα διαχείρισης της υποδομής SCADA ως προς την:

• Διαχείριση της συνδρομής (δηλ. Εγγραφή και κατάργηση εγγραφής)

στο μοντέλο αστοχίας που σχετίζεται με κάθε συγκεκριμένο τύπο εξοπλισμού.

• Βάση δεδομένων ερωτημάτων για μοντέλα αποτυχίας για εκείνα που ταιριάζουν με δεδομένο σύνολο ιδιοτήτων



Σχ.15 Αρχιτεκτονική πλατφόρμας προσαρμοστικών λειτουργιών (APO)

Όπως απεικονίζεται στο Σχ. 15, το AOP είναι δομημένο ως μορφή υπηρεσίας επιπέδων:

• Η Υποδομή Fog που περιλαμβάνει εξοπλισμό δικτύωσης με συγκεκριμένες δυνατότητες Fog και προβλέπει υπηρεσίες επικοινωνίας από άκρο σε άκρο.

• Το λειτουργικό σύστημα υποστήριξης (OSS) που αξιοποιεί την Υποδομή Ομίχλης για την παροχή των συνήθων λειτουργιών διαχείρισης περιουσιακών στοιχείων και υποστήριξης επιχειρήσεων (π.χ. απογραφή, συντήρηση, παροχή κλπ.)

Για να αξιοποιήσει αποτελεσματικά βασικά χαρακτηριστικά της Υποδομής Fog, το AOP περιλαμβάνει διάφορα λειτουργικά στοιχεία.

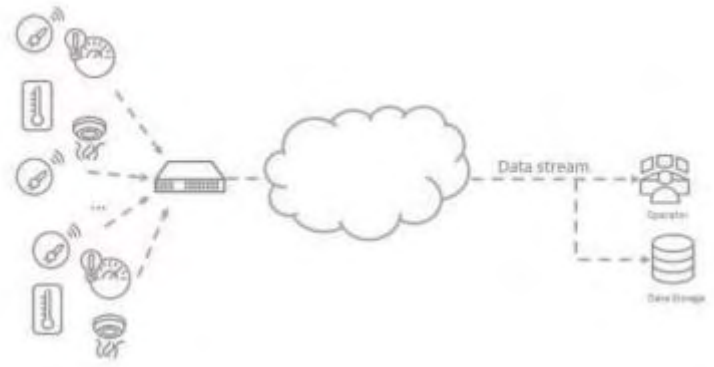
Στο σχήμα 15 φαίνονται τα βασικά συστατικά στοιχεία της αρχιτεκτονικής. Το λειτουργικό στοιχείο Model Building (MB) συνδυάζει στατικές πληροφορίες σχετικά με τα μοντέλα αστοχίας του εξοπλισμού μαζί με δυναμικές πληροφορίες που συγκεντρώνονται από την διάρκεια λειτουργίας του μοντέλου στη βιομηχανική εφαρμογή. Στα προηγούμενα δεδομένα υπάρχει πρόσβαση από τη βάση δεδομένων του μοντέλου αποτυχίας (απεικονίζεται ως μοντέλο DB στο Σχ. 15). που παρέχει μια διεπαφή στους κατασκευαστές για εγγραφή και ενημέρωση αυτών των πληροφοριών.

Στο λειτουργικό στοιχείο Rule Mapper (RM) έχει ανατεθεί χαρτογράφηση του συνταγμένου μοντέλου με ένα σύνολο κανόνων χειρισμού κυκλοφορίας κατανοητών από την υποδομή Software Defined Networking (SDN). Για κάθε συγκεκριμένο τύπο εξοπλισμού, το RM αλληλοεπιδρά με το MB μέσω της διεπαφής Iz (Σχ.15 για πρόσβαση στη κατάλληλη έκδοση του σχετικού συνταγμένου μοντέλου. Κανόνες χειρισμού κυκλοφορίας (δηλαδή, κανόνες DMo,Data in Motion) εξακολουθούν να υπάρχουν στη βάση δεδομένων κανόνων (απεικονίζονται ως Κανόνες DB στο Σχ. 15) υπό έλεγχο του στοιχείου RM.

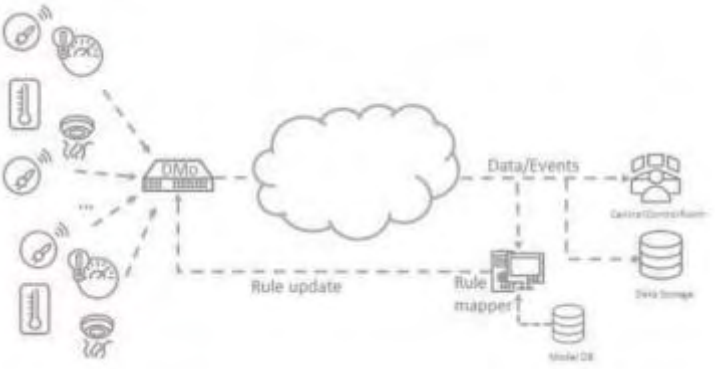
Μια βάση δεδομένων (απεικονίζεται ως DB υποδομής στο σχήμα 15) περιέχει περιγραφές των συσκευών που περιλαμβάνονται στην υποδομή, τη διάταξη ανάπτυξης και διασυνδέσεις αυτών (π.χ. τοπολογία δικτύωσης κ.λπ.), μαζί με τη σχετική διαμόρφωση και ρυθμίσεις παροχής. Οι κανόνες χειρισμού της κυκλοφορίας είναι προσβάσιμοι μέσω του στοιχείου RM μέσω της διεπαφής Iz που φαίνεται στο Σχ.15.

Σενάριο Εφαρμογής

Το σενάριό υπό εξέταση αποτελείται από Ν αισθητήρες που παρακολουθούν την κατάσταση των μηχανημάτων σε ανάπτυξη εντός του δικτύου. Κάθε αισθητήρας αναφέρει τακτικά τις καταγεγραμμένες τιμές μέσω ενός δρομολογητή σε έναν κεντρικό διακομιστή, όπως απεικονίζεται στο Σχ. 16. Οι μετρήσεις αποθηκεύονται σε μια βάση δεδομένων και απεικονίζονται σε έναν χειριστή. Ο στόχος της ανάπτυξης των αισθητήρων και η συγκέντρωση των μετρήσεων είναι η διερεύνηση πιθανών ανωμαλιών στα μηχανήματα που παρακολουθούνται. Το σενάριο που εξετάστηκε δοκιμάζεται χρησιμοποιώντας από τη μία πλευρά έναν κοινό δρομολογητή (Σχ.16Α) και από την άλλη έναν δρομολογητή / πύλη που υποστηρίζει DMo (Σχ.16Β), με σκοπό τη σύγκριση του κεντρικού σχήματος με την προσέγγιση APO.



Σχ.16Α Σενάριο υπό εξέταση AOP με χρήση απλού δρομολογητή



Σχ.16Β Σενάριο υπό εξέταση AOP με χρήση DMo δρομολογητή

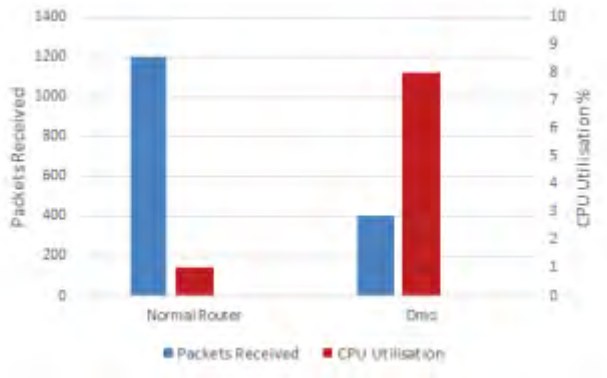
Εφαρμογή για μείωση Κυκλοφορίας Πακέτων Δεδομένων

Σε αυτή τη περίπτωση του σεναρίου εφαρμογής φαίνεται η μείωση της ποσότητας των δεδομένων που φτάνουν στο χειριστή (και αποθηκεύονται στο ιστορικό) χρησιμοποιώντας έναν δρομολογητή που υποστηρίζει DMo.

Το σενάριο απαρτίζεται από αισθητήρα θερμοκρασίας N μετρήσεων σε χρόνο t με τιμές στο διάστημα (50,70) κατά Gaussian κατανομή και τιμές εκτός αυτών καταγεγραμμένες ως ανωμαλίες. Προκειμένου να μειωθεί ο όγκος των αποθηκευμένων δεδομένων, είναι δυνατόν να εφαρμόσει η προσέγγιση AOP. Σε αυτήν την περίπτωση, ο διακομιστής εκτελεί αλγόριθμο Machine Learning (ML) (π.χ. k-Means) που είναι εκπαιδευμένος να κατανοήσει τη φυσιολογική συμπεριφορά της συσκευής. Στη συνέχεια, χρησιμοποιώντας το Rule Mapper (RM), ορίζεται ένας κανόνας για λήψη μόνο δεδομένων εκτός του φυσιολογικού εύρους για την αναγνώριση ανωμαλιών. Αυτά τα δεδομένα θα σταλούν ως συμβάντα στον χειριστή.

Στην περίπτωσή μας, το ML δείχνει ότι ο μέσος όρος των λαμβανομένων τιμών είναι κοντά στο 60, που αντιστοιχεί σε κανονική συμπεριφορά και στέλνει στο DMo έναν νέο κανόνα που καθορίζει ο δρομολογητής για προώθηση στον διακομιστή μόνο τιμών που αντιπροσωπεύουν ανωμαλίες, π.χ., στο εύρος [0, 50] και [70, 100]. Προφανώς,

το εύρος μπορεί να ρυθμιστεί διαφορετικά ανάλογα με τα όρια.



Σχ.17 Πλήθος Πακέτων και Χρήση Πόρων CPU σε APO διάταξη

Υπάρχει τεράστια μείωση στον αριθμό μεταδιδόμενων πακέτων. Το Σχ. 17, απεικονίζει επίσης τη χρήση της CPU του δρομολογητή. Παρατηρούμε ότι η χρήση DMo αξιώνει μόνο το 8% της CPU για μείωση στο 1/3 των εισερχομένων πακέτων.

Πολιτική Κατανομής Πόρων σε APO

Η πολιτική κατανομής πόρων σε μια APO εφαρμογή βιομηχανίας 4.0 είναι ιδιαίτερα ευέλικτη και πολύπλοκη. Με την χρήση routers με DMo έχουμε εισαγωγή προγραμματιζόμενης τεχνητής νοημοσύνης στους δρομολογητές οι οποίοι εκτελούν δυναμικές αποφάσεις σχετιζόμενες με την σωστή λειτουργία των συσκευών δικτύου IoT,FoG και ενημέρωση της κατάσταση τους σε πραγματικό χρόνο στους χειριστές.

Το δίκτυο μπορεί να προσαρμόζεται δυναμικά ώστε να αποκλείονται οι συσκευές που παρουσιάζουν ανωμαλίες στη λειτουργία τους και να ενημερώνεται άμεσα ο κατασκευαστής για μελλοντική αντικατάσταση τους . Έτσι δε χρειάζεται να υπάρξει χρόνος αργίας για συντήρηση των ελλαττωματικών αισθητήρων ή συσκευών.

Επομένως στο μοντέλο APO παρατηρείται ότι υπάρχει κύρια εστίαση πάνω στην δυνατότητα χρήσης δρομολογητών με ικανή επεξεργαστικής ισχύ και πόρους , με δυνατότητες λήψης αποφάσεων για βελτίωση της ταχύρρυθμης και υγιούς λειτουργίας του υπολογιστικού δικτύου ομίχλης.

2.4 ΣΥΓΚΡΙΣΗ ΑΛΓΟΡΙΘΜΩΝ ΚΑΤΑΝΟΜΗΣ ΠΟΡΩΝ ΚΑΙ ΕΠΙΛΟΓΗ ΓΙΑ ΠΡΟΣΟΜΟΙΩΣΗ

Οι 3 αλγόριθμοι-μοντέλα που αναφέραμε παρουσιάζουν ομοιότητες αλλά και σημαντικές διαφορές αν και όλα ικανοποιούν πλήρως τις βασικές παράμετρούς λειτουργίας κατασκευών FOG του πίνακα 1.

Συγκεκριμένα ο καθένας τους έχει διαφορετική εστίαση πάνω στην σύγχρονη αρχιτεκτονική ενός IoT-Edge-Fog δικτύου υπολογιστικής ομίχλης.

1. Ο αλγόριθμος VirtualFOG, εστιάζει στο υλικό μέρος της υλοποίησης και συγκεκριμένα στην βελτιστοποίησης της λειτουργίας των module και tuple της εφαρμογής υπολογιστικής ομίχλης μέσω εικονικοποίησης των διαθέσιμων πόρων των φυσικών συσκευών.
2. Ο αλγόριθμος PTPN εστιάζει στις ενότητες εφαρμογών και ροές των δεδομένων μέσω μαθηματικής αναλογίας με δίκτυα PetriNets ανάμεσα στις υλικές συσκευές.
3. Το μοντέλο APO εστιάζει στην βελτίωση του υπολογιστικού δικτύου με το προγραμματισμό συσκευών δρομολογητών που συνδέουν τις φυσικές οντότητες (π.χ. Edge-FoG-Cloud Nodes) , οι οποίοι παραλαμβάνουν το μεγαλύτερο βάρος χρήσης πόρων και επεξεργαστικής ισχύος για λειτουργίες επικοινωνίας από τις συσκευές FoG.

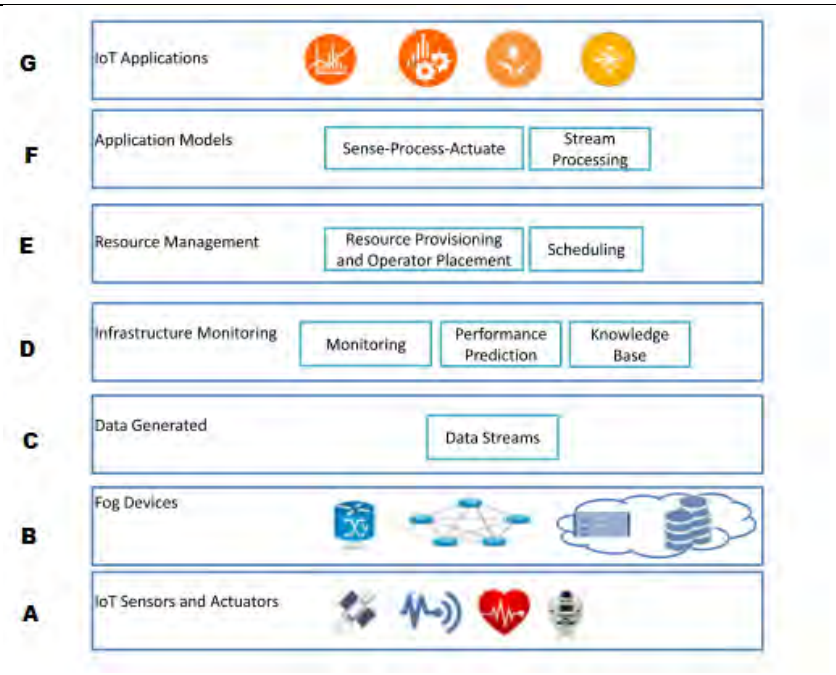
Στο επόμενο κεφάλαιο θα αναλύσουμε την βελτιστοποίηση ενός δικτύου βιομηχανίας 4.0 με διάταξη IoT-Edge-FoG με το προσομοιωτή iFogSim κάνοντας χρήση της λογικής των μοντέλων VirtualFog και Timed PetriNets. Η εικονικοποίηση των πόρων είναι εγγενής από το ίδιο το μοντέλο του CloudSim που βασίζεται και ο προσομοιωτής iFogSim και ενσωματώνεται στα επίπεδα παρακολούθησης και διαχείρισης πόρων της αρχιτεκτονικής του. Παράλληλα η λογική της σύνθεσης των εργασιών των PTPN που εκτελούνται στους πόρους υπολογιστικής ομίχλης στα FoG Nodes μπορεί να εισαχθεί εύκολα στο iFogSim με την κατάλληλη προσαρμογή των τιμών κόστους (cost values) και των tuple ανάμεσα στα module στο προγραμματισμό.

3 ΠΡΟΣΟΜΟΙΩΤΗΣ IFOGSIM

3.1 ΠΕΡΙΒΑΛΛΟΝ ΠΡΟΣΟΜΟΙΩΤΗ, ΛΕΙΤΟΥΡΓΙΕΣ ΚΑΙ ΔΥΝΑΤΟΤΗΤΕΣ

Η αρχιτεκτονική του περιβάλλοντος υπολογιστών Fog στο iFogSim (Σχ. 18) είναι πολλαπλών επιπέδων.

Κάθε υφιστάμενο επίπεδο είναι υπεύθυνο για συγκεκριμένες εργασίες για τη διευκόλυνση της λειτουργίας υψηλότερων επιπέδων. [7]



Σχ.18 Αρχιτεκτονική Fog Computing στο iFogSim

A.

Το κατώτερο στρώμα (A) περιλαμβάνει συσκευές IoT που είναι εκείνες που αλληλοεπιδρούν στον πραγματικό κόσμο και είναι η πηγή ή το απόθεμα δεδομένων (source and data sink).

Οι **αισθητήρες IoT** λειτουργούν ως πηγή δεδομένων για εφαρμογές και διανέμονται σε διαφορετικές γεωγραφικές τοποθεσίες, ανιχνεύοντας στο περιβάλλον και εκπέμποντας παρατηρούμενες τιμές στα ανώτερα στρώματα μέσω πυλών (gateways) για περαιτέρω επεξεργασία.

Οι **ενεργοποιητές IoT** λειτουργούν στο κάτω μέρος της αρχιτεκτονικής και είναι υπεύθυνοι για τον έλεγχο ενός μηχανισμού ή συστήματος. Οι ενεργοποιητές συνήθως σχεδιάζονται για να ανταποκρίνονται σε εντολές από εφαρμογές του ανώτερου επιπέδου οι οποίες δίνονται βάσει πληροφοριών που συλλαμβάνονται από τους αισθητήρες.

Κάθε συσκευή στο IoT είναι είτε πηγή είτε μια συλλογή δεδομένων(sink) και ως εκ τούτου μπορεί να μοντελοποιηθεί από έναν αισθητήρα ή έναν ενεργοποιητή.
Οι δημιουργοί της εφαρμογής iFogSim έχουν προσδιορίσει 5 τύπους τεχνολογιών συσκευών εισόδου οι οποίοι καταλέγονται στο Στρώμα A

Αισθητήρες, Έξυπνους Αναγνώστες, Κάμερες, Μικρόφωνα και Συλλέκτες.

Καθεμία από αυτές τις πρότυπες συσκευές έχει ορισμένα χαρακτηριστικά εκπομπής δεδομένων, για παράδειγμα, χρόνο διακοπής ή μέγεθος των τμημάτων δεδομένων που εκπέμπονται. (*intermission size, size of data chunk*)

Με αυτό το τρόπο στο iFogSim ένας αισθητήρας συνδέεται με ορισμένα χαρακτηριστικά εκπομπής δεδομένων, τα οποία μπορούν να προσαρμοστούν για την προσομοίωση οποιουδήποτε είδους συσκευής IoT εκπομπής δεδομένων, που κυμαίνεται από έξυπνες κάμερες και περιβαλλοντικούς αισθητήρες έως wearables και κινητά τηλέφωνα.

Καθώς το iFogSim δεν δύναται να προσομοιώσει άμεσα φυσικά γεγονότα δικτύου χαμηλού επιπέδου, όπως η διαχείριση παρεμβολών μεταξύ πυκνών συστοιχιών συσκευών IoT, οι χρήστες πρέπει να ανάγουν αυτά τα ζητήματα χαμηλού επιπέδου σε χαρακτηριστικά υψηλού επιπέδου όπως καθυστέρηση ή περιορισμένο εύρος ζώνης σύνδεσης μεταξύ των συσκευών και των αντίστοιχων πυλών IoT. Η διεξοδική

δημιουργία διαφορετικών προφίλ παραμετροποίησης μπορεί να επιτρέψει στον χρήστη να δημιουργήσει ένα ρεαλιστικό μοντέλο φυσικής συμπεριφοράς ασύρματου επιπέδου το οποίο μπορεί να φορτωθεί κάθε φορά στο iFogSim και να χρησιμοποιηθεί για διαφορετικές περιπτώσεις.

B.

Το στρώμα (B) περιλαμβάνει **συσκευές FOG** δηλαδή οποιοδήποτε στοιχείο στο δίκτυο που μπορεί να φιλοξενήσει λειτουργικές μονάδες εφαρμογών.

Συσκευές FOG που συνδέουν αισθητήρες στο δίκτυο ονομάζονται γενικά πύλες. Οι συσκευές FOG περιλαμβάνουν επίσης πόρους Cloud που παρέχονται κατά απαίτηση από γεωγραφικά κατανεμημένα κέντρα δεδομένων.

Οι συσκευές τακτοποιούνται σε μια ιεραρχική τοπολογία με άμεση επικοινωνία δυνατή μόνο μεταξύ ενός ζευγαριού γονέα-παιδιού στην ιεραρχία. Μια μονάδα εφαρμογής που λειτουργεί σε μια συσκευή FOG είναι υπεύθυνη για επεξεργασία όλων των δεδομένων που παράγονται από στοιχεία κάτω από τη συσκευή στην ιεραρχία.

Το iFogSim βασίζεται στον ορισμό του Fog Computing που τον παρουσιάζει ως υποδομή με παρόμοια χαρακτηριστικά με το cloud με μόνη διαφορά ότι οι υπολογιστές είναι τοποθετημένοι κοντά στην άκρη του δικτύου(near EDGE,IoT). Δεν υποστηρίζει την επικοινωνία μεταξύ συσκευών (device to device) καθώς προϋποθέτει μια ιεραρχική οργάνωση συσκευών ομίχλης. Η τοποθέτηση της εφαρμογής πραγματοποιείται με κατεύθυνση Πάνω προς Κάτω και δεν είναι δυνατή (προς το παρόν) η εκφόρτωση modules σε άλλη συσκευή στο ίδιο επίπεδο ιεραρχίας. Ως εκ τούτου, σενάρια όπως smartphone to smartphone είναι αδύνατα προς το παρών αλλά μελλοντικά θα υπάρξει και πρόβλεψη για αλλαγή αυτής της περιοριστικής ιεραρχικής δομής στην επικοινωνία μεταξύ των συσκευών.

C.

Οι ροές δεδομένων IoT, οι οποίες είναι ακολουθίες τιμών (αναφέρονται ως **tuple** στο iFogSim),σχηματίζουν το επίπεδο (C) στην αρχιτεκτονική. Αυτές οι ροές μπορούν να εκπέμπονται από αισθητήρες (στην περίπτωση αυτή περιέχουν ανεπεξέργαστα δεδομένα) ή μπορούν να μεταδοθούν από ένα module εφαρμογής σε άλλο ή ακόμη και από μονάδα (module) της εφαρμογής σε ενεργοποιητές.

Οι ροές δεδομένων δημιουργούνται επίσης από συσκευές FOG, με τη μορφή λεπτομερειών χρήσης πόρων, οι οποίες υποβάλλονται σε επεξεργασία από το επίπεδο παρακολούθησης (D) για την απόκτηση πληροφοριών σχετικά με την κατάσταση των συσκευών.

D.

Στο **επίπεδο παρακολούθησης** (D) γίνεται εποπτεία της χρήσης πόρων, της κατανάλωσης ενέργειας και της διαθεσιμότητας αισθητήρων, ενεργοποιητών και συσκευών FOG.

Τα στοιχεία παρακολούθησης παρέχουν αυτές τις πληροφορίες στο επίπεδο διαχείρισης πόρων (E) και μπορούν να τις παρέχουν σε άλλες υπηρεσίες ως απαιτείται. Για λόγους απλότητας του iFogSim, σχετικά περίπλοκα στοιχεία επιπέδου παρακολούθησης όπως η πρόβλεψη απόδοσης και η βάση γνώσεων δεν έχουν συμπεριληφθεί ακόμη. Αυτά τα στοιχεία μπορούν, ωστόσο, να προκύψουν έμμεσα από *εγγραφές* που προκύπτουν από επεξεργασία στατιστικών στοιχείων χρήσης πόρων και μηνύματα διαθεσιμότητας που εκπέμπονται από όλες τις συσκευές FOG αντίστοιχα.

E.

Η **διαχείριση πόρων** (E) είναι το βασικότερο συστατικό της αρχιτεκτονικής του iFogSim. Αποτελείται από στοιχεία που διαχειρίζονται συνεκτικά τους πόρους του επιπέδου συσκευών FOG με τέτοιο τρόπο ώστε να ικανοποιούνται οι περιορισμοί

αξιοπιστίας υπηρεσιών QoS σε επίπεδο εφαρμογής και να ελαχιστοποιείται η σπατάλη τους.

Τα 2 επιμέρους στοιχεία αυτού του επιπέδου είναι :

α) Διαθεσιμότητα Πόρων και Τοποθέτηση operator

b) Δρομολογητής (Scheduler)

Έχουμε την παρακολούθηση της κατάστασης των διαθέσιμων FOG assets (πληροφορίες παρέχονται από την υπηρεσία παρακολούθησης) για τον προσδιορισμό των καλύτερων υποψηφίων συσκευών για τη φιλοξενία μιας μονάδας εφαρμογής και την κατανομή των πόρων στα modules της συσκευής.

Αυτό το επίπεδο λειτουργεί με βάση τις απαιτήσεις ποιότητας των υπηρεσιών που θέτει το επίπεδο εφαρμογής F.

Η πολιτική διαχείρισης πόρων μπορεί να είναι αρκετά περίπλοκη ώστε να επιτρέπει τη μετεγκατάσταση στοιχείων και δυναμικές αλλαγές στην κατανομή πόρων συσκευών σε στοιχεία FOG.

Επιπλέον, η εφαρμογή του επιπέδου διαχείρισης πόρων μπορεί να διανεμηθεί (με κάθε συσκευή να διαχειρίζεται τους δικούς της πόρους χωρίς παγκόσμια γνώση) ή να συγκεντρώνεται (με όλες οι συσκευές που στέλνουν πληροφορίες πόρου σε έναν κεντρικό διαχειριστή πόρων) ή ένα υβρίδιο και των δύο. Η τρέχουσα έκδοση του iFogSim, ωστόσο, παρέχει μια στατική πολιτική τοποθέτησης εφαρμογών - με τις λειτουργικές μονάδες να κατανέμονται στατικά σε συσκευές FOG.

Αυτή η πολιτική μπορεί να αντικατασταθεί από δυναμικές πολιτικές που μπορούν να μετ’εγκαταστήσουν μονάδες (modules) σε άλλες συσκευές FOG με βάση κριτήρια όπως η ενεργειακή κατανάλωση και καθυστέρηση.

F.

Μοντέλα εφαρμογής (προγραμματισμός). Οι εφαρμογές που αναπτύχθηκαν για εκτελούνται στις συσκευές FOG βασίζονται στα δεδομένα κατανεμημένης ροής (distributed data flow, *DDF* model)

Μια εφαρμογή μοντελοποιείται ως συλλογή ενοτήτων(modules) , οι οποίες αποτελούν τα στοιχεία επεξεργασίας δεδομένων. Τα δεδομένα εξόδου μιας μονάδας i (module) μπορεί να είναι δεδομένα εισόδου σε μια άλλη μονάδα j. Έτσι δημιουργείται αλληλεξάρτηση δεδομένων i,j

Αυτό το μοντέλο εφαρμογής επιτρέπει την **παράσταση εφαρμογής σε μορφή κατευθυνόμενου γραφήματος.**

Στο γράφημα αυτό οι κορυφές είναι τα modules των εφαρμογών και οι κατευθυνόμενες ακμές αναπαριστούν την ροή των δεδομένων ανάμεσα στις μονάδες αυτές.

G.

Δύο από τους κύριους μοχλούς δημιουργίας της υπολογιστικής ομίχλης υπήρξαν η ανάγκη ανταπόκρισης σε πραγματικό χρόνο και ταυτόχρονα η ανάγκη αυξημένης επεκτασιμότητας με τον πολλαπλασιασμό του IoT.

Οι εφαρμογές IoT τείνουν να έχουν αισθητήρες ως πηγές δεδομένων, οι οποίοι βρίσκονται συχνά σε μορφή πλειάδων. Η προτεινόμενη αρχιτεκτονική iFogSim υποστηρίζει δύο μοντέλα που χρησιμοποιούνται για εφαρμογές IoT.

1. Μοντέλο Sense-Process-Actuate

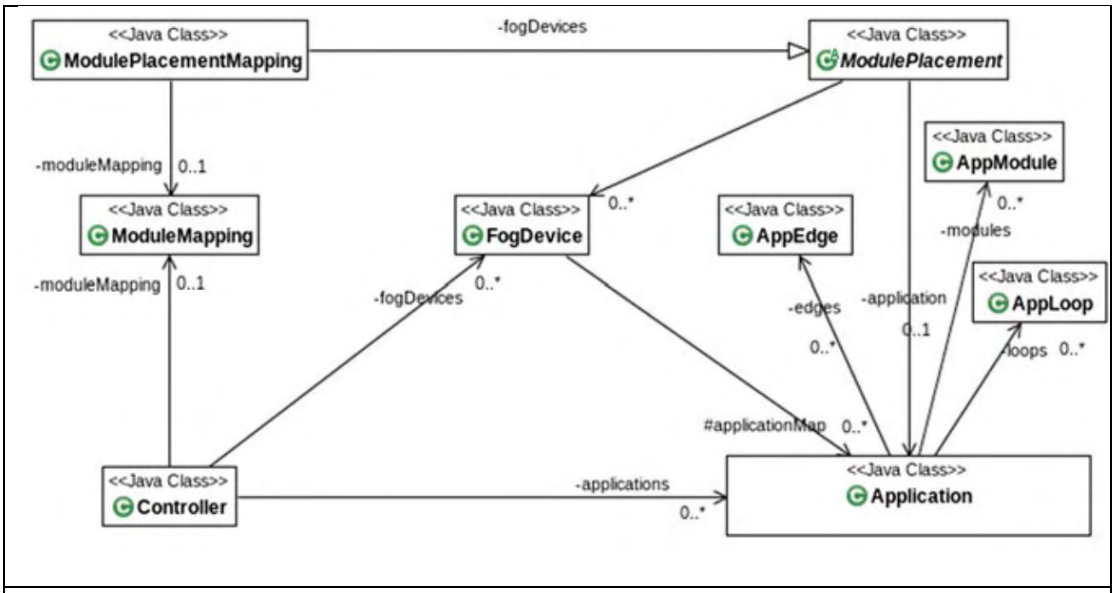
Οι πληροφορίες που συλλέγονται από τους αισθητήρες μεταδίδονται ως ροές δεδομένων, οι οποίες τροφοδοτούνται για επεξεργασία σε εφαρμογές που εκτελούνται σε συσκευές ομίχλης. Οι εντολές που προκύπτουν από την επεξεργασία, αποστέλλονται σε ενεργοποιητές.

2. Μοντέλο επεξεργασίας ροής.

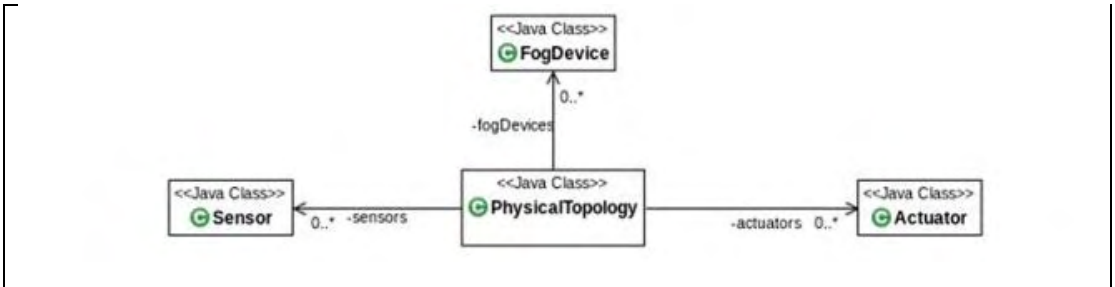
Το μοντέλο επεξεργασίας ροής διαθέτει ένα δίκτυο λειτουργικών μονάδων που εκτελούνται σε συσκευές ομίχλης που επεξεργάζεται συνεχώς ροές δεδομένων που εκπέμπονται από αισθητήρες. Οι πληροφορίες που εξ ορύσσονται από τις εισερχόμενες ροές αποθηκεύονται σε κέντρα δεδομένων (data centers) για μεγάλης κλίμακας και μακροπρόθεσμη ανάλυση.

Για την εφαρμογή λειτουργιών της αρχιτεκτονικής iFogSim, χρησιμοποιούνται λειτουργίες προσομοίωσης βασικών συμβάντων που υπάρχουν στο CloudSim.

Οι οντότητες στο CloudSim, όπως τα κέντρα δεδομένων, επικοινωνούν μεταξύ τους με λειτουργίες μετάδοσης μηνυμάτων (αποστολή συμβάντων). Ως εκ τούτου, το βασικό επίπεδο λειτουργίας του CloudSim είναι υπεύθυνο για τον χειρισμό συμβάντων μεταξύ των υπολογιστικών στοιχείων Fog στο iFogSim. Οι κύριες τάξεις του iFogSim απεικονίζονται στα σχήματα 19 και 20.



Σχ.19 Θεμελιώδεις Κλάσεις στο iFogSim



Σχ.20 iFogSim Κλάσεις Φυσικής Τοπολογίας

Η υλοποίηση του iFogSim αποτελείται από προσομοιωμένες οντότητες και υπηρεσίες. Τα στοιχεία της αρχιτεκτονικής μοντελοποιούνται ως κλάσεις Java στο iFogSim. Στη συνέχεια θα περιγραφεί η λειτουργία και η αλληλεπίδραση αυτών των κλάσεων.

Θεμελιώδεις Κλάσεις στο iFogSim

- **FogDevice** (συσκευή Fog)

Αυτή η κλάση καθορίζει **τα χαρακτηριστικά υλικού** (hardware characteristics) των συσκευών ομίχλης και τις συνδέσεις τους με άλλες συσκευές FOG, αισθητήρες και ενεργοποιητές. Έχοντας υλοποιηθεί με επέκταση από την κλάση PowerDatacenter στο CloudSim, τα κύρια χαρακτηριστικά του FogDevice είναι **διαθεσιμότητα σε : μνήμη, επεξεργαστή, μέγεθος χώρου αποθήκευσης, εύρους ζώνης uplink και downlink** . Οι μέθοδοι σε αυτήν την κλάση καθορίζουν τον τρόπο δρομολόγησης και χρήσης των πόρων μιας συσκευής FOG από τις λειτουργικές μονάδες εφαρμογών που εκτελούνται σε αυτή τη συσκευή και πώς αναπτύσσονται και απενεργοποιούνται modules αυτής. Παράκαμψη αυτών των μεθόδων επιτρέπει στους προγραμματιστές να προσθέσουν προσαρμοσμένες πολιτικές για τις παραπάνω λειτουργίες.

- **Sensor** (αισθητήρας)

Οι κλάσεις sensor είναι οντότητες που λειτουργούν όπως οι αισθητήρες IoT που περιγράφονται στην αρχιτεκτονική. Η κλάση περιέχει τιμές που αντιπροσωπεύουν τα τεχνικά χαρακτηριστικά ενός αισθητήρα, που κυμαίνονται από τη συνδεσιμότητά του έως τα χαρακτηριστικά εξόδου. Η κλάση περιέχει μια τιμή ως **χαρακτηριστικό αναφοράς στη συσκευή ομίχλης πύλης** με την οποία είναι συνδεδεμένος ο αισθητήρας και **την καθυστέρηση σύνδεσης μεταξύ τους.**

Το πιο σημαντικό, καθορίζει τα χαρακτηριστικά εξόδου του αισθητήρα και την κατανομή της μετάδοσης tuple, η οποία προσδιορίζει το ρυθμό άφιξης tuple στην πύλη. Ορίζοντας τις κατάλληλες τιμές αυτών των χαρακτηριστικών, συσκευές όπως έξυπνες κάμερες μπορούν να προσομοιωθούν.

- **Actuator** (ενεργοποιητής)

Αυτή η κλάση αντιπροσωπεύει έναν ενεργοποιητή καθορίζοντας το αποτέλεσμα της ενεργοποίησης και τις ιδιότητες σύνδεσης δικτύου. Η κλάση ορίζει μια **μέθοδο για την εκτέλεση μιας ενέργειας κατά την άφιξη μιας πλειάδας (tuple)** από μια λειτουργική μονάδα εφαρμογής, η οποία μπορεί να παρακαμφθεί για την εφαρμογή προσαρμοσμένων εφέ ενεργοποίησης. Ένα attribute στην κλάση αναφέρεται στην πύλη στην οποία είναι συνδεδεμένος ο ενεργοποιητής και τον λανθάνον χρόνο αυτής της σύνδεσης.

- **Tuple** (πλειάδα ή ακολουθία πληροφοριών)

Η Tuple αποτελεί τη **θεμελιώδη μονάδα επικοινωνίας** μεταξύ οντοτήτων στην υπολογιστική ομίχλη και πραγματοποιεί **το επίπεδο ροής δεδομένων** στην αρχιτεκτονική. Εμφανίζεται ως οντότητα τάξης tuple στο iFogSim, το οποίο κληρονομείται από την κλάση Cloudlet του CloudSim. Μια πλειάδα πληροφοριών χαρακτηρίζεται από τον τύπο της και τις ενότητες (modules) εφαρμογών προέλευσης και προορισμού. Τα χαρακτηριστικά της κλάσης καθορίζουν τις απαιτήσεις σε επεξεργασία εκφρασμένα σε MI (Million Instructions) και το μήκος της πληροφορίας που εμπεριέχεται στην ακολουθία.

- **Application** (εφαρμογή)

Ο σχεδιασμός εφαρμογών στο iFogSim ακολουθεί το **μοντέλο DDF**, στο οποίο μια εφαρμογή διαμορφώνεται ως **κατευθυνόμενο γράφημα.**

Οι κορυφές του **κατευθυνόμενου ακυκλικού γραφήματος (DAG)** αντιπροσωπεύουν μονάδες που εκτελούν επεξεργασία σε εισερχόμενα δεδομένα και οι ακμές υποδηλώνουν εξαρτήσεις δεδομένων μεταξύ ενοτήτων (modules).

Αυτές οι οντότητες πραγματοποιούνται χρησιμοποιώντας τις ακόλουθες κατηγορίες.

1. **AppModule**

Οι εμφανίσεις της κλάσης AppModule αντιπροσωπεύουν στοιχεία επεξεργασίας εφαρμογών υπολογιστικής ομίχλης και υλοποιούν τις κορυφές του DAG στο μοντέλο DDF.

Ο AppModule δημιουργείται επεκτείνοντας την κλάση PowerVm στο CloudSim Για κάθε εισερχόμενο tuple, μια παρουσία AppModule το επεξεργάζεται και δημιουργεί πλειάδες εξόδου που αποστέλλονται σε επόμενες ενότητες στο DAG.

Ο αριθμός πλειάδων εξόδου ανά πλειάδα εισόδου αποφασίζεται χρησιμοποιώντας ένα μοντέλο επιλεκτικότητας , το οποίο μπορεί να βασίζεται σε κλασματική επιλεκτικότητα ή ένα εκρηκτικό μοντέλο.

2. **AppEdge**

Μια παρουσία AppEdge υποδηλώνει την εξάρτηση δεδομένων μεταξύ ενός ζεύγους λειτουργικών μονάδων και αντιπροσωπεύει ένα κατευθυνόμενο άκρο στο μοντέλο εφαρμογής DDF. Κάθε άκρη χαρακτηρίζεται από τον τύπο της πλειάδας που μεταφέρει, ο οποίος συλλαμβάνεται από το χαρακτηριστικό tupleType της κλάσης AppEdge μαζί με τις απαιτήσεις επεξεργασίας και το μήκος των δεδομένων που μεταφέρουν ως πληροφορία αυτές οι πλειάδες. Το iFogSim υποστηρίζει δύο τύπους εφαρμογών άκρων - **περιοδικά και βάσει συμβάντων**.

Οι ακολουθίες πληροφορίας σε ένα AppEdge με εφαρμογές περιοδικών συμβάντων εκπέμπονται σε τακτά χρονικά διαστήματα.

Μια πλειάδα σε ένα άκρο με εφαρμογές βάσει συμβάντων **E=(u,v)** αποστέλλεται όταν η μονάδα πηγής **u** λαμβάνει ένα tuple και το μοντέλο επιλεκτικότητας του u επιτρέπει την εκπομπή πλειάδων e.

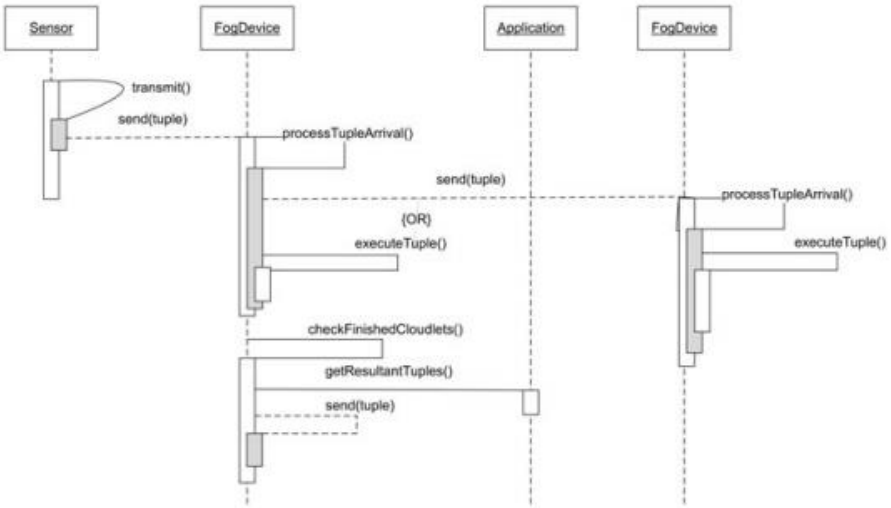
3. AppLoop

Το AppLoop είναι μια επιπλέον κλάση, που χρησιμοποιείται για τον καθορισμό των βρόχων ελέγχου-διαδικασίας που ενδιαφέρουν τον χρήστη**.** Στο iFogSim,ο προγραμματιστής μπορεί να καθορίσει τους βρόχους ελέγχου για τη μέτρηση της καθυστέρησης από άκρο σε άκρο. Μια εγγραφή AppLoop είναι ουσιαστικά μια λίστα των modules που ξεκινούν από την προέλευση του βρόχου έως τη μονάδα όπου τελειώνει ο βρόχος.

Ένα διάγραμμα ροής που δείχνει την εκπομπή πλειάδων και την επακόλουθη εκτέλεση φαίνεται στο Σχήμα 21. Μια ακολουθία δεδομένων δημιουργείται από ένα αισθητήρα και αποστέλλεται στην πύλη με την οποία είναι συνδεδεμένος ο αισθητήρας.

Η συνάρτηση callback για το χειρισμό μιας εισερχόμενης tuple ***processTupleArrival()*** καλείται μόλις η πλειάδα φτάσει στη συσκευή ομίχλης (πύλη). Σε περίπτωση που το tuple πρέπει να δρομολογηθεί σε άλλη συσκευή Fog, αυτό αποστέλλεται αμέσως χωρίς επεξεργασία. Διαφορετικά, εάν τοποθετηθεί η μονάδα εφαρμογής στην οποία πρέπει να εκτελεστεί η πλειάδα στη συσκευή λήψης ομίχλης η πλειάδα υποβάλλεται άμεσα για εκτέλεση.

Η συνάρτηση ***checkCloudletCompletion ()*** καλείται στην συσκευή ομίχλης μετά την ολοκλήρωση της εκτέλεσης της πλειάδας.



Σχ.21 Διάγραμμα Ροής Πηγής και Επεξεργασίας Tuple στο iFogSim

Εκτός από τις βασικές λειτουργίες επεξεργασίας tuple, οι προσομοιωμένες υπηρεσίες που διατίθενται στο iFogSim είναι οι εξής:

- **Monitoring Service** (υπηρεσία παρακολούθησης)

Στην τρέχουσα έκδοση του iFogSim, κάθε συσκευή παρακολουθεί και διατηρεί τα τρέχοντα στατιστικά στοιχεία χρήσης πόρων.

Η μέθοδος ***executeTuple ()*** στην κλάση FogDevice περιέχει τη λογική επεξεργασίας πλειάδων όπου η συσκευή ενημερώνει συνεχώς για τους διαθέσιμους πόρους της. **Αυτά τα στατιστικά στοιχεία μπορούν επίσης να ενθυλακωθούν σε μια πλειάδα και να σταλούν στο επίπεδο διαχείρισης πόρων για τρέξιμο σχετικά με τη χρήση πολιτικών διαχείρισης πόρων.**

Τέτοιες πληροφορίες μπορεί να είναι χρήσιμες στον χρήστη για τη μελέτη της απόδοσης της εφαρμογής στην υποδομή υπολογιστικής ομίχλης και μπορούν να ληφθούν ως αρχεία καταγραφής για ανάλυση εκτός σύνδεσης.

Ωστόσο, **η τρέχουσα έκδοση του προσομοιωτή δεν εμφανίζει τις τιμές ακατέργαστης χρήσης στον χρήστη.** Αυτές οι τιμές χρήσης πόρων εισάγονται σε ένα

σχετικό μοντέλο ισχύος για τον υπολογισμό της κατανάλωσης ισχύος της συσκευής, η οποία αναφέρεται στο τέλος της προσομοίωσης. Κάθε συσκευή ομίχλης (FogDevice) σχετίζεται με ένα μοντέλο ισχύος (π.χ. PowerModelLinear), το οποίο εκτιμά την κατανάλωση ενέργειας σε μια δεδομένη χρήση της CPU.

- **Resource Management Service**

  (υπηρεσία διαχείρισης πόρων)

Το iFogSim διαθέτει δύο επίπεδα διαχείρισης πόρων για εφαρμογές

τοποθέτηση modules και προγραμματισμός — οι οποίοι απομονώνονται ως ξεχωριστές πολιτικές για τη διευκόλυνση της επέκτασης και της προσαρμογής του δικτύου υπολογιστικής ομίχλης.

1. **Application placement** (αίτησης τοποθέτησης εφαρμογης)

   Η πολιτική τοποθέτησης καθορίζει τον τρόπο τοποθέτησης των λειτουργικών μονάδων σε όλες τις συσκευές Fog με υποβολή αίτησης. Η διαδικασία τοποθέτησης μπορεί να καθοδηγείται από στόχους όπως η ελαχιστοποίηση του λανθάνοντος χρόνου από άκρο σε άκρο, χρήση δικτύου, λειτουργικό κόστος ή κατανάλωση ενέργειας. Η κλάση **ModulePlacement** είναι η αφηρημένη πολιτική τοποθέτησης που πρέπει να επεκταθεί για την ενσωμάτωση νέων πολιτικών.

2. **Application scheduling** (προγραμματισμός εφαρμογών)

   Ο προγραμματισμός χρήσης πόρων της host συσκευής ομίχλης σε λειτουργικές μονάδες αποτελεί το δεύτερο επίπεδο της διαχείρισης πόρων. Ο προεπιλεγμένος προγραμματιστής πόρων κατανέμει εξίσου τους πόρους μιας συσκευής μεταξύ όλων των ενεργών ενοτήτων εφαρμογών. Η πολιτική προγραμματισμού εφαρμογών μπορεί να προσαρμοστεί παρακάμπτοντας τη μέθοδο ***updateAllocatedMips*** εντός της κλάσης ***FogDevice.***

Ιδιότητες κοινές με CloudSim

Το iFogSim έχει εφαρμοστεί ως επέκταση του CloudSim και ως εκ τούτου κληρονομεί μια σειρά από σημαντικές δυνατότητες του CloudSim.

- **Ταυτόχρονη εκτέλεση πολλαπλών εφαρμογών**

Ο iFogSim επιτρέπει την εκτέλεση πολλαπλών εφαρμογών στην υποδομή την ίδια στιγμή. Κάθε εφαρμογή θα πρέπει να δημιουργήσει ξεχωριστούς αισθητήρες / ενεργοποιητές και ένα μοντέλο τοποθέτησης. λειτουργικών μονάδων στο δίκτυο υπολογιστικής ομίχλης. Ο χρήστης μπορεί επίσης να κωδικοποιήσει διαφορετική συμπεριφορά για διαφορετικές εφαρμογές στην πολιτική τοποθέτησης, επιτρέποντας έτσι διακριτές πολιτικές τοποθέτησης λειτουργικών μονάδων για συγκεκριμένες εφαρμογές.

- **Μετεγκατάσταση των ενοτήτων εφαρμογών**

Το iFogSim υποστηρίζει τη μετεγκατάσταση των λειτουργικών μονάδων από τη μια συσκευή ομίχλης στην άλλη, επειδή το AppModule είναι απλώς μια επέκταση της κλάσης VM του CloudSim.

Οι συσκευές ομίχλης που εμπλέκονται στη μετεγκατάσταση πρέπει να εκτελεστούν σχετική τήρηση αρχείου για την καταγραφή της μετεγκατάστασης και την άμεση κυκλοφορία στη μονάδα μετεγκατάστασης.

- **Πολιτικές διαχείρισης πόρων με δυνατότητα σύνδεσης**

Οι πολιτικές διαχείρισης των πόρων στο iFogSim μπορούν να εφαρμοστούν ως pluggable ενότητες που αποσυνδέονται από την προσομοίωση. Αυτό επιτρέπει τον εύκολο επαναλαμβανόμενο πειραματισμό του ίδιου σεναρίου με πολλαπλές πολιτικές διαχείρισης πόρων.

Ενσωματωμένες τακτικές τοποθέτησης modules

Το iFogSim έρχεται εγκατεστημένο με δύο τεχνικές τοποθέτησης λειτουργικής μονάδας

A) *τοποθέτηση μόνο σε Cloud*

B) *τοποθέτηση κοντά στα άκρα δικτύου (Edge-Ward placement).*

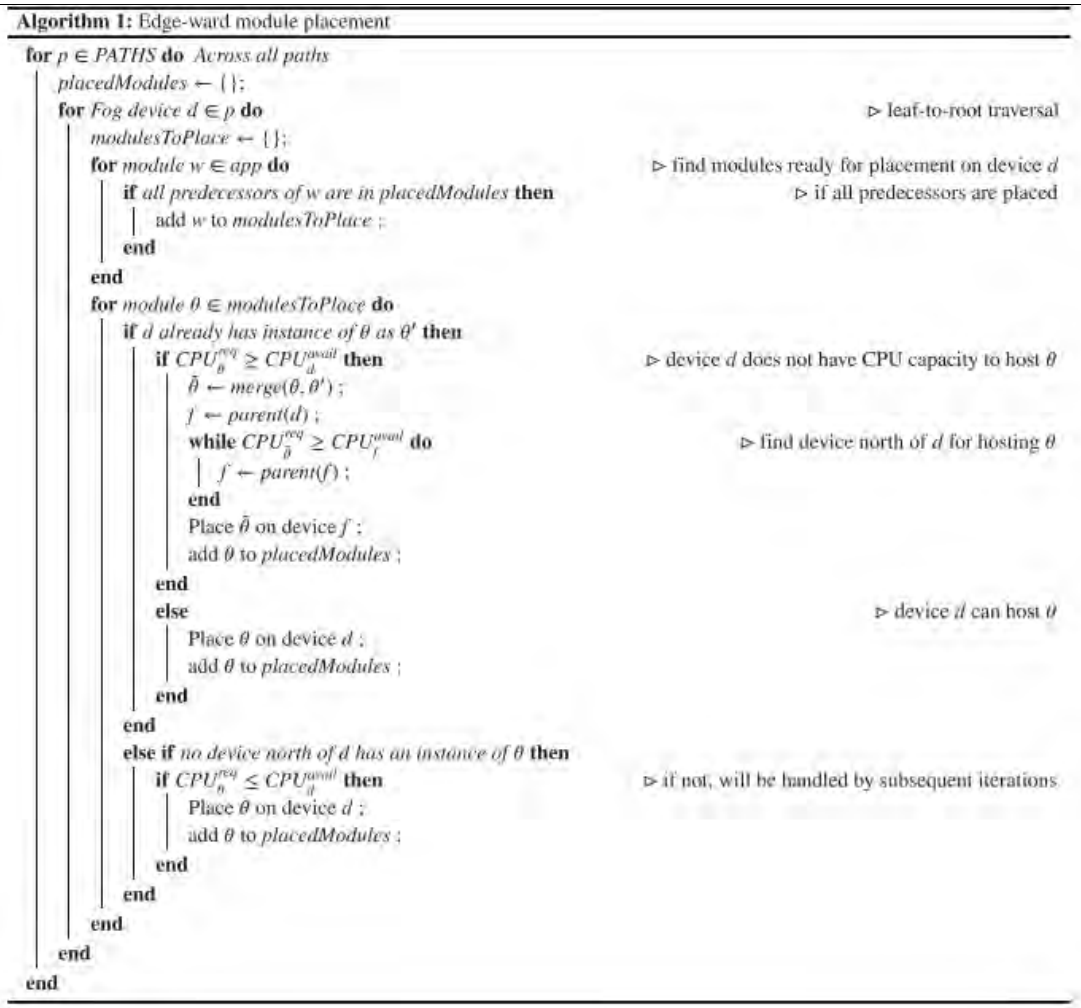
A) Πρόκειται για την κλασική προσέγγιση τεχνολογιών CLOUD κατά την οποία όλες οι λειτουργικές μονάδες μιας εφαρμογής εκτελούνται σε κέντρα δεδομένων. Ο βρόχος λειτουργίας sense-process-actuate σε τέτοιες εφαρμογές υλοποιείται με την ύπαρξη αισθητήρων που μεταδίδουν αισθητήρια δεδομένα στο CLOUD όπου υποβάλλονται σε επεξεργασία και οι ενεργοποιητές ενημερώνονται εάν απαιτείται δράση.

B) Η στρατηγική τοποθέτησης Edge-Ward ευνοεί την ανάπτυξη λειτουργικών μονάδων κοντά στην άκρη του δίκτυού. Ωστόσο, οι συσκευές που βρίσκονται κοντά στην άκρη του δικτύου - όπως οι δρομολογητές και τα σημεία πρόσβασης - ενδέχεται να μην είναι αρκετά ισχυρά από άποψη χαρακτηριστικών υπολογιστικής ισχύος (memory,processing power,etc.) για να φιλοξενήσουν όλους τους χειριστές (operators) της εφαρμογής. Σε μια τέτοια κατάσταση, η στρατηγική επαναλαμβάνεται σε συσκευές ομίχλης προς το cloud με κατεύθυνση από κάτω προς τα πάνω και προσπαθεί να τοποθετήσει τους υπόλοιπους χειριστές σε εναλλακτικές συσκευές.

Αυτή η στρατηγική τοποθέτησης modules τόσο στα άκρα δικτύου όσο και στις υπόλοιπες συσκευές φαίνεται στο παρακάτω **αλγόριθμο** (σχ.22).

Χαρακτηριστικό του αλγορίθμου αυτού είναι η αλληλεπίδραση Fog και Cloud στο οικοσύστημα του υπολογιστικού δικτύου. Για λόγους απλότητας εισάγονται όροι North , South (βόρεια και νότια) ως αντιστοιχία προαγωγής η υποβίβασης αντίστοιχα των Nodes στο IoT-EDGE-FOG-CLOUD δίκτυο

προσομοίωσης με κατεύθυνση από πάνω προς τα κάτω. Άλλος ένα παραλληλισμός είναι αυτός της ρίζας – (Cloud ή proxy server) και των φύλλων (Edge devices).



Algorithm 1: Edge-ward module placement

for $p \in PATHS$ **do** *Across all paths*
  $placedModules \leftarrow \{\}$;
  for *Fog device* $d \in p$ **do** ▷ leaf-to-root traversal
    $modulesToPlace \leftarrow \{\}$;
    for *module* $w \in app$ **do** ▷ find modules ready for placement on device d
      if *all predecessors of* w *are in placedModules* **then** ▷ if all predecessors are placed
        add w to $modulesToPlace$;
      end
    end
    for *module* $\theta \in modulesToPlace$ **do**
      if d *already has instance of* θ *as* θ' **then**
        if $CPU_{\theta}^{req} \geq CPU_{d}^{avail}$ **then** ▷ device d does not have CPU capacity to host θ
          $\tilde{\theta} \leftarrow merge(\theta, \theta')$;
          $f \leftarrow parent(d)$;
          while $CPU_{\tilde{\theta}}^{req} \geq CPU_{f}^{avail}$ **do** ▷ find device north of d for hosting θ
            $f \leftarrow parent(f)$;
          end
          Place $\tilde{\theta}$ on device f ;
          add θ to $placedModules$;
        end
        else ▷ device d can host θ
          Place θ on device d ;
          add θ to $placedModules$;
        end
      end
      else if *no device north of* d *has an instance of* θ **then**
        if $CPU_{\theta}^{req} \leq CPU_{d}^{avail}$ **then** ▷ if not, will be handled by subsequent iterations
          Place θ on device d ;
          add θ to $placedModules$;
        end
      end
    end
  end
end

Σχ.22 Edge-Ward Αλγόριθμος τοποθέτησης modules στο iFogSim

Παρατηρήσεις Edge Ward Placement

Ο αλγόριθμος τοποθέτησης Edge-Ward επαναλαμβάνει όλες τις διαδρομές από φύλλο σε ρίζα στην τοπολογία του φυσικού δικτύου και τοποθετεί ενότητες σε κάθε τέτοια διαδρομή. Για κάθε διαδρομή, ο αλγόριθμος επαναλαμβάνεται από το φύλλο (συσκευή στο νότιο άκρο) - τυπικά συσκευή EDGE– έως τη ρίζα (βορειότερη συσκευή) - συνήθως το CLOUD - και σταδιακά τοποθετεί τις ενότητες σε αυτές.

Για κάθε συσκευή σε μια διαδρομή, προσδιορίζονται οι ενότητες που μπορούν να τοποθετηθούν σε αυτό. Μια μονάδα θ μπορεί να τοποθετηθεί σε μια συσκευή d μόνο εάν όλες οι άλλες ενότητες που θα πρέπει να τοποθετηθούν στα νότια του θ (οι προκάτοχοι), έχουν ήδη τοποθετηθεί στο ίδιο φύλλο από τη ρίζα. Μόλις οι ενότητες που πρόκειται να τοποθετηθούν, καθοριστούν, ο αλγόριθμος προσπαθεί να τις τοποθετήσει μία προς μία στη συσκευή.

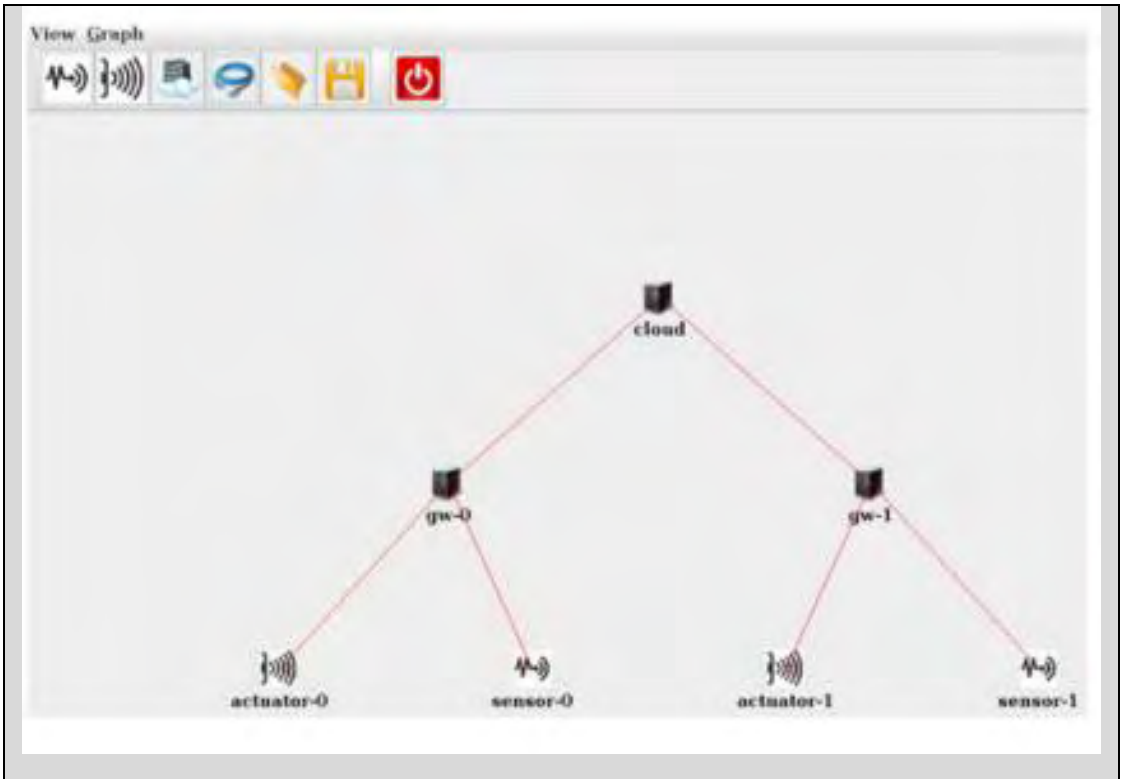
Παραδείγματος χάρη, για μονάδα θ που τοποθετείται στη συσκευή d. Ο αλγόριθμος ελέγχει πρώτα εάν μια παρουσία του θ είναι ήδη τοποθετημένη στο d (αν είναι δηλαδή μέρος κάποιας άλλης διαδρομής από φύλλο σε ρίζα). Εάν ναι, αυτές οι παρουσίες συγχωνεύονται και τοποθετούνται στο d, αν μπορεί να φιλοξενήσει τις συγχωνευμένες παρουσίες. Διαφορετικά, ο αλγόριθμος αναζητά συσκευές βόρεια του d για να τοποθετήσει τη συγχωνευμένη παρουσία. Ωστόσο, εάν ούτε η συσκευή d ούτε οποιαδήποτε συσκευή βόρεια του d έχει παρουσία θ η μονάδα (module) τοποθετείται στο d. Σε περίπτωση που είχε οποιαδήποτε συσκευή βόρεια του d μια περίπτωση θ, αυτή θα διαμορφωθεί από τα πρώιμα βήματα του αλγόριθμου σε μια μελλοντική εκτέλεση του βρόχου που επαναλαμβάνεται.

Επειδή το iFogSim υποθέτει ότι μια μονάδα m τοποθετημένη στη συσκευή d θα χειριστεί όλες τις εισερχόμενες πλειάδες από συσκευές νότια του d στην ιεραρχία, πολλαπλές εμφανίσεις μιας ενότητας συγχωνεύονται και τοποθετούνται βόρεια από αυτήν κατά αυτή τη στρατηγική τοποθέτησης. Παραδείγματος χάρη για συσκευές dp και dc όπου το dp είναι ο γονέας του dc στην τοπολογία και οι δύο φιλοξενούν την ίδια ενότητα m. Σε μια τέτοια περίπτωση, οποιαδήποτε εισερχόμενη κίνηση για την ενότητα m στο dc θα υποβληθεί σε επεξεργασία εκεί και δεν θα σταλεί βόρεια.

Γραφικό περιβάλλον διεπαφής χρήστη

Για να διευκολυνθεί η περιγραφή της τοπολογίας του φυσικού δικτύου, έχει δημιουργηθεί ένα Γραφικό περιβάλλον διεπαφής χρήστη (Graphical User Interface) πάνω από τη λογική της εφαρμογής iFogSim. Το GUI επιτρέπει στο χρήστη να σχεδιάσει φυσικά στοιχεία όπως συσκευές ομίχλης, αισθητήρες, ενεργοποιητές και συνδέσμους σύνδεσης. Πάνω σε όλους τις επιμέρους αυτές οντότητες του δικτύου ο χρήστης μπορεί να ορίσει άμεσα όλα τα φυσικά μεγέθη – χαρακτηριστικά τους. Οι σχεδιαζόμενες τοπολογίες μπορούν να αποθηκευτούν και να φορτωθούν από μονάδα αποθήκευσης άμεσα και είναι σε μορφή αρχείου JSON.

Οι φυσικές τοπολογίες μπορούν να δημιουργηθούν μέσω GUI (FogGui.java) και μέσω προγραμματισμού API της Java. Το Σχήμα 23 δείχνει ένα δείγμα φυσικής τοπολογίας, που περιλαμβάνει έναν αισθητήρα, μια πύλη, μια εικονική μηχανή cloud και τις συνδέσεις. Η αναπαράσταση JSON αυτής της φυσικής τοπολογίας φαίνεται στο Σχήμα 24.



Σχ.23 iFogSim GUI

Σχ.24 Αρχείο JSON τοπολογίας δικτύου

ΠΡΟΣΕΓΓΙΣΗ ΓΙΑ ΤΗΝ ΠΡΟΣΟΜΟΙΩΣΗ ΠΕΡΙΒΑΛΛΟΝΤΩΝ IoT ΚΑΙ ΤΟΝ ΕΛΕΓΧΟ ΤΕΧΝΙΚΩΝ ΔΙΑΧΕΙΡΙΣΗΣ ΠΟΡΩΝ

Τα γενικά 3 βήματα που πρέπει να ακολουθηθούν για την εκτέλεση μια προσομοίωσης IoT-Edge-FoG στο iFogSim και για τον έλεγχο τεχνικών διαχείρισης πόρων είναι τα εξής :

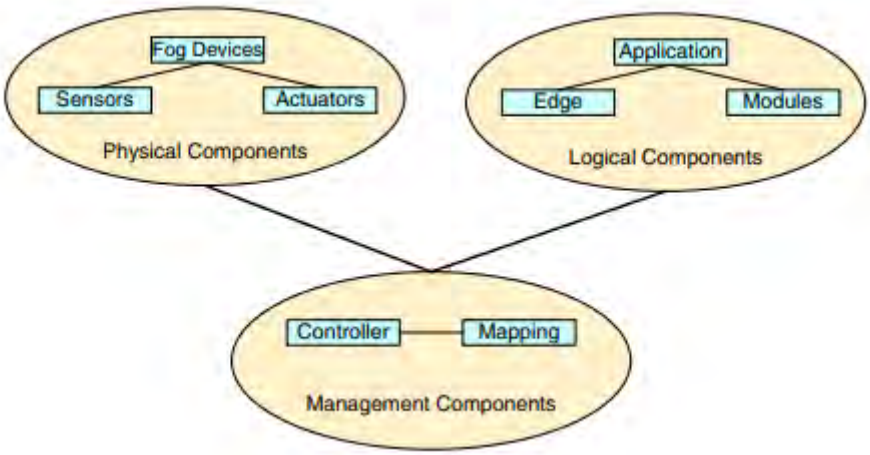
1) Να δημιουργηθούν φυσικές οντότητες και να καθοριστούν οι δυνατότητες και οι διαμορφώσεις τους. Αυτά περιλαμβάνουν αισθητήρες, πύλες και εικονικές μηχανές cloud, καθώς και τους συνδέσμους που περιγράφουν τον τρόπο σύνδεσης αυτών των οντοτήτων. Όπως αναφέρθηκε προηγουμένως, αυτό επιτυγχάνεται είτε χρησιμοποιώντας το GUI είτε μέσω προγραμματισμού χρησιμοποιώντας τις προαναφερθείσες κλάσεις iFogSim.

Για να προσομοιωθεί το μοντέλο του φόρτου εργασίας στο δίκτυο μας πρέπει πρώτα να οριστούν οι ρυθμοί μετάδοσης Tuple των αισθητήρων χρησιμοποιώντας το ***transmitDistribution*** (ένα χαρακτηριστικό στην Java κλάση ***Sensor***)

2) Να γίνει το μοντέλο των εφαρμογών που θα εκτελούνται στο δίκτυο υπολογιστικής ομίχλης. Όπως εξηγήθηκε νωρίτερα, μια εφαρμογή διαμορφώνεται ως DAG και κατασκευάζεται μέσω 3 κλάσεων ***AppModule, AppEdge και AppLoop***

3) Τέλος, πρέπει να καθορίσουμε **πολιτικές τοποθέτησης και προγραμματισμού** που χαρτογραφούν τις ενότητες εφαρμογών σε συσκευές ομίχλης.

Οι πολιτικές ενδέχεται να λαμβάνουν υπόψη εύρος κριτηρίων, συμπεριλαμβανομένης της καθυστέρησης επεξεργασίας από άκρο σε άκρο, της απόδοσης, του κόστους, της κατανάλωσης ενέργειας και ειδικούς περιορισμούς επι μέρους συσκευών. Οι κλάσεις ***ModulePlacement*** και ***Controller*** είναι αυτές όπου βρίσκεται η λογική τοποθέτησης που εφαρμόζεται.



Σχ.25 Αρχιτεκτονική Υλοποίησης Fog Computing στο iFogSim

3.2 ΕΠΑΓΓΕΛΜΑΤΙΚΕΣ ΕΦΑΡΜΟΓΕΣ ΜΕ ΒΑΣΗ ΤΟΝ IFOGSIM

Ο προσομοιωτής iFogSim έχει εκτεταμένη χρήση σε εφαρμογές δεντρικής διάταξης IoT-Fog Computing. Παρουσιάζει εκτεταμένη υποστήριξη σε μεταβαλλόμενες σε μέγεθος από άκρο σε άκρο τοπολογίες καθώς και πληθώρα επιλογών τοποθέτησης modules και προγραμματισμού αυτών με βάση διάφορα τεχνικά χαρακτηριστικά όπως απόκριση , ενέργεια ή κόστος.

1) Εφαρμογές Intelligent Surveillance through Distributed Camera Networks

Το κατανεμημένο σύστημα παρακολούθησης με κάμερες έχει εκτεταμένη εφαρμογή τα τελευταία χρόνια, σε ευρύ φάσμα δραστηριοτήτων όπως η δημόσια ασφάλεια και η ασφάλεια στην κατασκευή, στη μεταφορά και στο τομέα υγείας. Ως εκ τούτου, χρειαζόμαστε εργαλεία που αναλύουν αυτόματα δεδομένα που προέρχονται από κάμερες και συνοψίζουν τα αποτελέσματα με τρόπο που είναι επωφελής για τον τελικό χρήστη. Κύρια χαρακτηριστικά ενός τέτοιου IoT-Fog δικτύου είναι:

Α) *Επικοινωνία χαμηλής καθυστέρησης (low latency)*

Για αποτελεσματική κάλυψη αντικειμένων, χρειάζονται οι παράμετροι Pan-Tilt-Zoom (PTZ) πολλαπλών καμερών για συντονισμό σε πραγματικό χρόνο με βάση την εικόνα που τραβήχτηκε.Η διαχείριση πρέπει να είναι σε πραγματικό χρόνο.

Β) *Διαχείριση μεγάλου όγκου δεδομένων.*

Οι βιντεοκάμερες στέλνουν συνεχώς καταγεγραμμένα καρέ βίντεο για επεξεργασία, γεγονός που προκαλεί τεράστια κυκλοφορία. Είναι απαραίτητο να επεξεργάζεται

τεράστιος όγκος δεδομένων στις συσκευές FoG χωρίς να φτάνει το δίκτυο σε κατάσταση συμφόρησης.

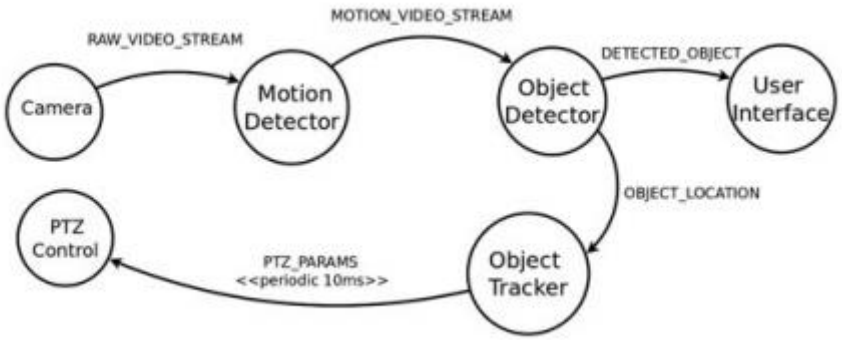
Γ) *Βαριά επεξεργασία σε βάθος χρόνου*

Η στρατηγική ελέγχου κάμερας πρέπει να ενημερώνεται συνεχώς έτσι ώστε να μαθαίνει τη βέλτιστη σάρωση PTZ. Αυτό απαιτεί ανάλυση των αποφάσεων που λαμβάνονται από τη στρατηγική ελέγχου για μεγάλο χρονικό διάστημα, το οποίο κάνει αυτήν την ανάλυση υπολογιστικά εντατική μακροπρόθεσμα.[7]

Μοντέλο εφαρμογής στο iFogSim

Όπως απεικονίζεται στο Σχήμα 26, η εφαρμογή Intelligent Surveillance αποτελείται από 5 κύριες ενότητες που εκτελούν επεξεργασία - Ανιχνευτής κίνησης, Ανιχνευτής αντικειμένων, Παρακολούθηση αντικειμένων, Έλεγχος PTZ και Διεπαφή χρήστη.

Η εφαρμογή παρακολούθησης τροφοδοτείται από ζωντανές ροές βίντεο από διάφορες κάμερες CCTV και ο έλεγχος PTZ σε κάθε κάμερα προσαρμόζει συνεχώς τις παραμέτρους PTZ.



Σχ.26 Μοντέλο εφαρμογής Intelligent Surveillance

Tuple type	CPU length	N/W length
RAW_ VIDEO_ STREAM	1000	20 000
MOTION_ VIDEO_ STREAM	2000	2000
DETECTED_ OBJECT	500	2000
OBJECT_ LOCATION	1000	100
PTZ_ PARAMS	100	100

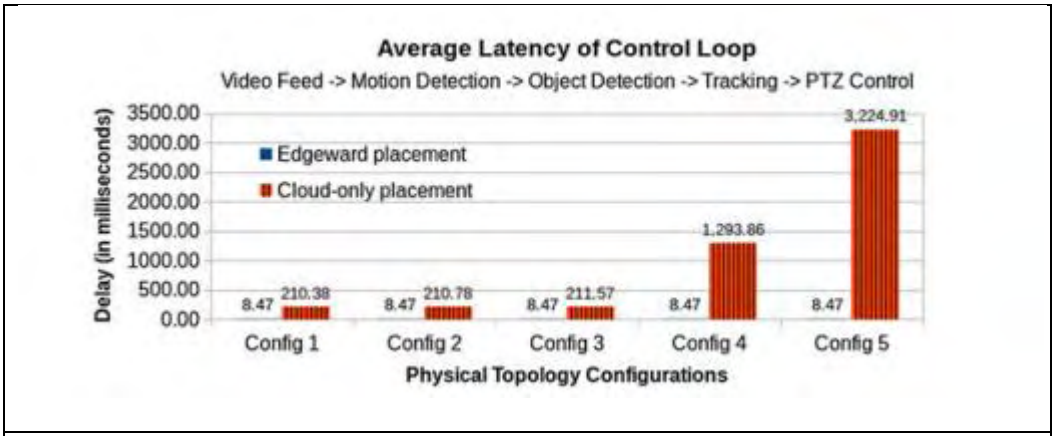
Πιν.2 Μορφές Tuple iFogSim Intelligent Surveillance

CPU length	NW length	Average interarrival time
1000 million instructions	20 000 bytes	5 ms

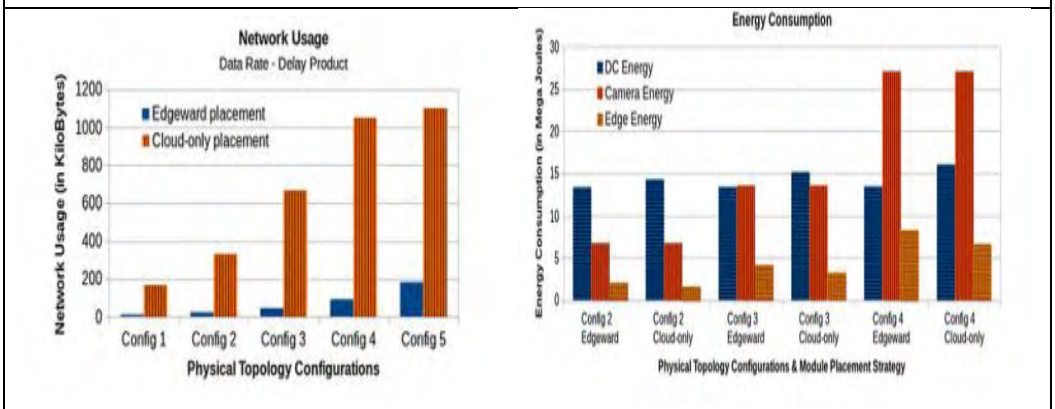
Πιν.3 Χαρακτηριστικά Διαμόρφωσης Καμερών ως Αισθητήρες iFogSim Intelligent Surveillance

Τα μοντέλα εφαρμογών, οι εξαρτήσεις δεδομένων και ο κύκλος ελέγχου ενδιαφέροντος είναι μοντελοποιημένα χρησιμοποιώντας τάξεις AppModule, AppEdge και AppLoop, αντίστοιχα. Οι ιδιότητες των πλειάδων (tuples) που μεταφέρονται από τα άκρα μεταξύ των ενοτήτων στην εφαρμογή περιγράφονται στον Πίνακα 2 .Ο Πίνακας 3 δείχνει τη διαμόρφωση των αισθητήρων που εμπλέκονται στην εφαρμογή. Η φυσική τοπολογία είναι 4 Fog Devices και οι κάμερες ως IoT devices. Είναι μοντελοποιημένη στο iFogSim μέσω τάξεων FogDevice, Sensor, PhysicalTopology και Actuator.

Αποτελέσματα Προσομοίωσης



Σχ.27 Μέση καθυστέρηση κύκλου ελέγχου



Σχ.28 Χρήση Δικτύου και Κατανάλωση Ενέργειας Προσομοίωσης Intelligent Surveillance

Με βάση τα παραπάνω διαγράμματα είναι κατανοητός ο ρόλος της βελτιστοποίησης της ταχύτητας και της μείωσης του κόστους όπως κατέδειξε η προσομοίωση iFogSim σε edge ward τοποθέτηση. Και στα 5 διαφορετικά σενάρια 1, 2 , 4 , 8 , 16 περιοχές εποπτείας αντιστοίχως των Config1, Config2, Config3, Config4, Config5 παρατηρείται εμφανής βελτίωση στη ταχύτητα απόκρισης, στη μείωση χρήσης δικτύου και στη κατανάλωση ενέργειας.

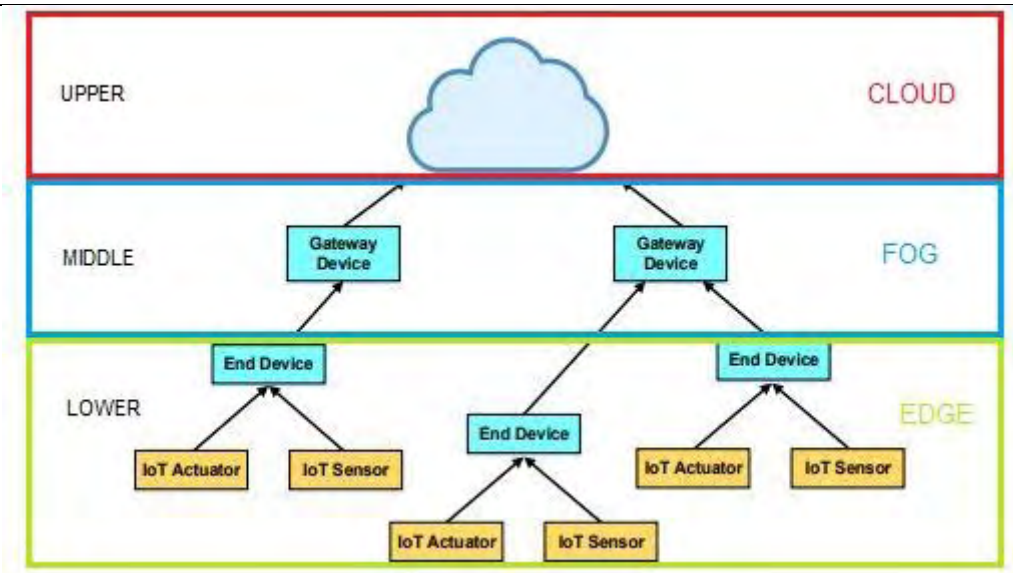
Εκτός της παραπάνω εμπορικής εφαρμογής αυτοματοποιημένης εποπτείας και ασφάλειας με την χρήση δικτύου καμερών εφαρμογές έχουμε και στο τομέας της υγείας άλλα και στην αυτοματοποίηση στην βιομηχανία 4.0, σενάριο το οποίο θα αναλυθεί στη συνέχεια.

3.3 ΠΑΡΑΜΕΤΡΟΠΟΙΗΣΕΙΣ ΣΕΝΑΡΙΟΥ ΕΚΤΕΛΕΣΗΣ

Αρχικά θα πρέπει να διαμορφωθεί η γενική τοπολογία ενός IoT-EDGE-FOG-CLOUD δικτύου Βιομηχανίας 4.0 όπως του Σχ.1. Στο iFogSim έχουμε αναπαράσταση μόνο αισθητήρων-ενεργοποιητών (χαμηλού επιπέδου συσκευές χωρίς δυνατότητες αποθήκευσης ή επεξεργασίας) και συσκευών FOG διαφορετικών επιπέδων.

Για αυτό θα οριστούν 3 διαφορετικά επίπεδα FOG Nodes.Στο χαμηλότερο επίπεδο (2) βρίσκονται συσκευές EDGE. Στο μεσαίο επίπεδο (1) FoG Servers σε τοπικό επίπεδο. Δύο αρχικά με επέκταση για περισσότερους αργότερα (Σχ.29). Τέλος, στο ανώτερο επίπεδο (0) βρίσκεται το CLOUD προσομοιωμένο στο iFogSim ως FOG Node ανώτατου επιπέδου (UPPER Level).

Οι συσκευές EDGE για λόγους απλότητας θα είναι smartphones με ενσωματωμένο αισθητήρα που στέλνει δεδομένα ροής (tuple) και την οθόνη τους ως ενεργοποιητή. Τα gateways είναι ασύρματου τύπου routers συνδεδεμένα με τις συσκευές FoG του μεσαίου επιπέδου. Άρα τα Fog Nodes του μεσαίου επιπέδου αποτελούν ουσιαστικά συσκευές ομίχλης πύλης (Gateway Fog Devices). Οι συσκευές ομίχλης πύλης γεφυρώνουν το κέντρο δεδομένων του cloud και τις τερματικές συσκευές ομίχλης (EDGE) για την εκτέλεση μιας αρθρωτής εφαρμογής. Για απλότητα, οι FoG συσκευές σε ίδια ιεραρχικά επίπεδα θεωρούνται ομοιογενείς. Η αισθητήρια συχνότητα λειτουργίας (sensing frequency) είναι ίδια για όλους τους αισθητήρες.



Σχ.29 Τοπολογία Δικτύου Σεναρίου Προσομοίωσης IIoT

Κύρια Χαρακτηριστικά IoT-Edge-Fog Δικτύου Προσομοίωσης

Όπως αναλύθηκε στο προηγούμενο σενάριο Έξυπνου Δικτύου Παρακολούθησης με πολλαπλό σύστημα καμερών οι κύριες προτεραιότητες του δικτύου ήταν ο χαμηλός χρόνος απόκρισης PTZ,η διαχείριση μεγάλου όγκου δεδομένων και η δυνατότητα βαρείας επεξεργασίας στους κόμβου FoG ανωτέρων επιπέδων. Στο υπό εξέταση σενάριο υλοποίησης IoT-Edge-FoG Βιομηχανίας 4.0 τα κύρια χαρακτηριστικά διαφοροποιούνται και είναι:

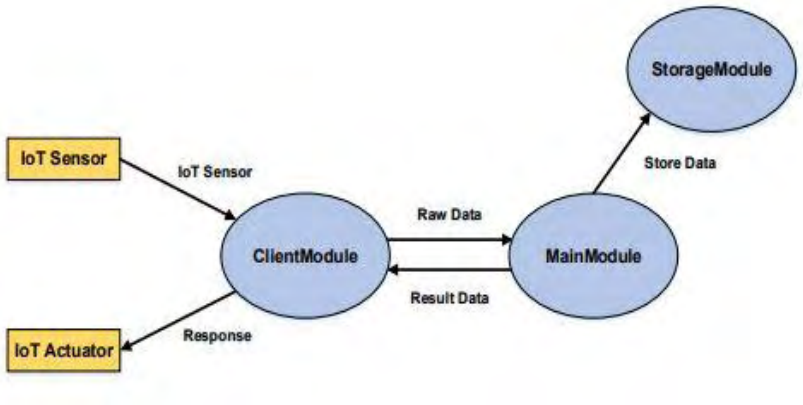
1) Δυνατότητα γρήγορης Επέκτασης του δικτύου με προσθήκη συσκευών IoT στο κατώτερο επίπεδο (Smartphones ή άλλα Edge Devices με ενσωμάτωση IoT αισθητήρων-ενεργοποιητών)

2) Ελαχιστοποίηση του κόστους της χρήσης του Cloud με διατήρηση του client module distribution σε Edge Nodes.

3) Ελαχιστοποίηση της χρήσης των πόρων επεξεργασίας στους κόμβους FoG του μεσαίου επιπέδου για τη βελτίωση της ταχύτητας του δικτύου αφού ο κύριος ρόλος τους θα είναι ως φορείς FoG Gateways.

Μοντέλο Εφαρμογής Σεναρίου Εκτέλεσης (Application Model)

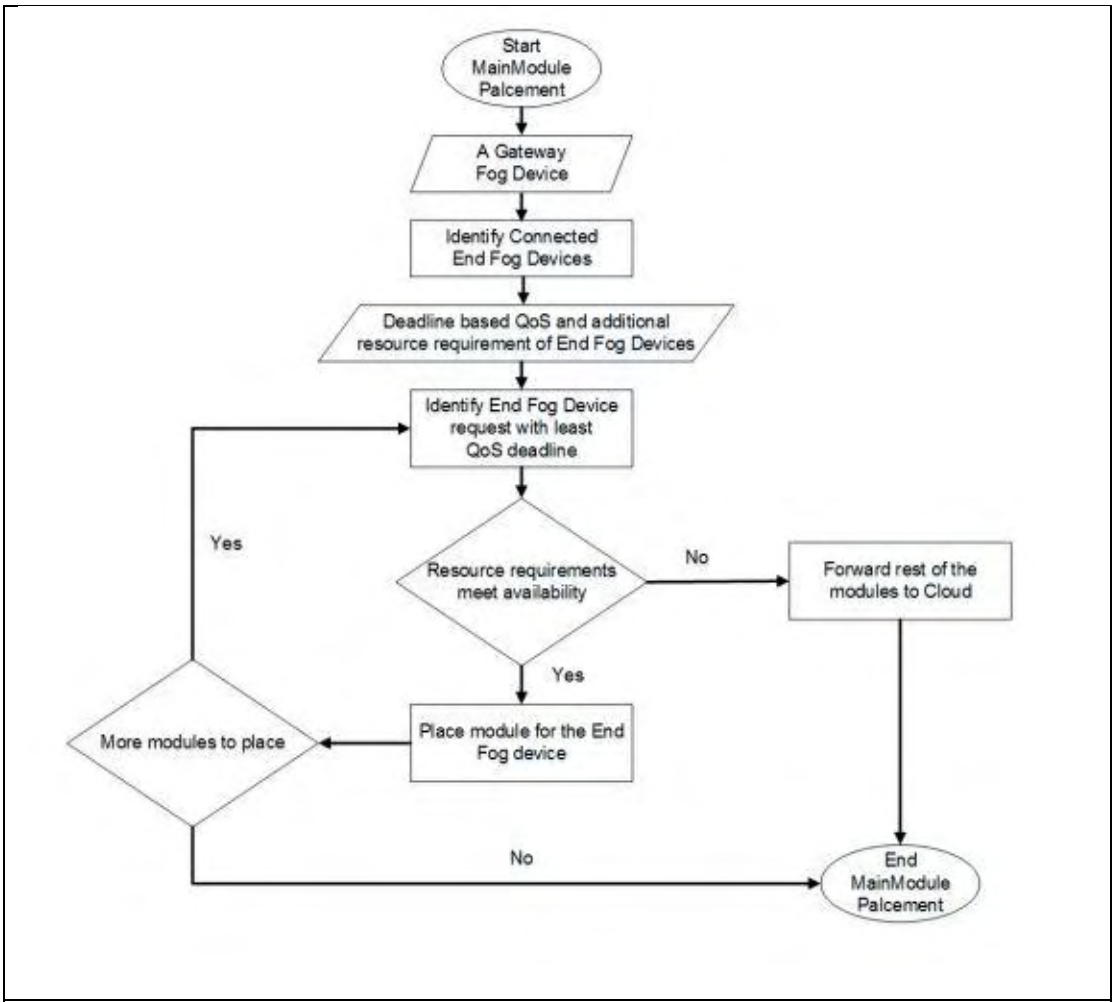
Το μοντέλο εφαρμογής απεικονίζεται στο Σχήμα 30. Το ClientModule τοποθετείται σε συσκευές EDGE και το StorageModule τοποθετείται στο CLOUD. Το MainModule απαιτεί ορισμένο αριθμό υπολογιστικών πόρων για να εκκινήσει και είναι εγκατεστημένο στις συσκευές Fog μεσαίου επιπέδου. Για να εξυπηρετηθεί η απαίτηση διαφορετικών συσκευών EDGE εντός προθεσμίας, επιπλέον πόροι μπορούν να ζητηθούν από τελικές συσκευές σε συνδεδεμένη πύλη με συσκευές ομίχλης μεσαίου επιπέδου.



Σχ.30 Μοντέλο Εφαρμογής Σεναρίου Εκτέλεσης

Πολιτική Διαχείρισης (Application Placement)

Στόχος του σεναρίου εκτέλεσης είναι η σωστή εξυπηρέτηση των MainApplication modules στις συσκευές FoG για διαφορετικό αριθμό End Devices(π.χ. smartphones) βάσει της προθεσμίας τους και της διαθεσιμότητας πόρων στις κεντρικές συσκευές. Για ευκολότερη κατανόηση, το διάγραμμα ροής της πολιτικής τοποθέτησης της εφαρμογής παρουσιάζεται στο Σχήμα 31.



Σχ.31 Πολιτική Τοποθέτησης Εφαρμογών Σεναρίου Εκτέλεσης

4. ΠΛΑΙΣΙΟ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ, ΣΕΝΑΡΙΑ ΥΛΟΠΟΙΗΣΗΣ

Για την εφαρμογή του επιλεχθέντος μοντέλου στο προσομοιωτή έγινε προσαρμογή στις κλάσεις .java στο ΠΑΡΑΡΤΗΜΑ Ι.

TestApplication (Main Class) , MyApplication, MyFogDevice, MyController, MySensor, MyActuator, MyModulePlacement, MyPlacement.

Σκοπός της προσομοίωσης με βάση την παραμετροποίηση είναι να εκτελεστεί για δεδομένο κύκλο εκτέλεσης (Control Loop) διαμοιρασμός client modules στα Edge devices και όταν υπάρξει πλήρωση των διαθέσιμων πόρων προώθηση του υπόλοιπου των λειτουργικών μονάδων στο Cloud.

Στα παραδείγματα εκτέλεσης θα εξετάσουμε **4 σενάρια λειτουργίας** με **Στοιχεία G και E**. Οπού G (Gateway Fog Devices) ο αριθμός των συσκευών Fog μεσαίου επιπέδου και E (Edge Devices) ο αριθμός τερματικών συσκευών Smartphones ανά Fog Device G.

4.1 ΒΗΜΑΤΑ ΛΕΙΤΟΥΡΓΙΑΣ – ΈΛΕΓΧΟΣ ΛΟΓΙΚΗΣ (CONTROLLER)

Τα βήματα του ελεγκτή λειτουργίας του προγράμματος προσομοίωσης είναι:

Α) Εκκίνηση του Main Module στο αρχικό Fog Device επιπέδου 1 όταν φτάσει σε ένα threshold η ροή δεδομένων tuple που ξεκινάει στους αισθητήρες των smartphones.

Β) Διαμοιρασμός Client Modules στα Edge Devices κάτω του αντίστοιχου FoG Device με την πολιτική τοποθέτησης Root-Leaf που ενσωματώνει ο iFogSim.

Γ) Διαμοιρασμός Client modules στα υπόλοιπα κλαδιά δεξιά του αρχικού. Ο αλγόριθμος επαναλαμβάνεται από το φύλλο (συσκευή στο νότιο άκρο) - τυπικά συσκευή EDGE– έως τη ρίζα (βορειότερη συσκευή) - συνήθως το CLOUD - και σταδιακά τοποθετεί τις ενότητες σε αυτές.

Δ) Τερματισμός της προσομοίωσης όταν ικανοποιηθεί μια εκ των 2 αρχικών συνθηκών του βρόχου πολιτικής τοποθέτησης.

4.2 ΦΥΣΙΚΕΣ ΤΙΜΕΣ – ΠΑΡΑΜΕΤΡΟΙ ΣΕΝΑΡΙΩΝ ΠΡΟΣΟΜΟΙΩΣΗΣ

Χαρακτηριστικές Τιμές Φυσικών Οντοτήτων Σεναρίου Εκτέλεσης

Layer	Network	Level	MIPS	RAM	UpBw	DownBw	UpLatency
UPPER	CLOUD	0	44800	40000	100	10000	1
MIDDLE	FOG	1	2800	4000	10000	10000	5
LOWER	EDGE	2	3200	1000	1000	10000	2

Πιν.4Α Παράμετροι Σεναρίου Εκτέλεσης (Resource Parameters)

Layer	Network	Level	RATE/MIPS	Busy Power(D)	Idle Power(D)
UPPER	CLOUD	0	0.01	16*103	16*83.25
MIDDLE	FOG	1	0.0	107.339	83.4333
LOWER	EDGE	2	0.0	87.53,	82.44

Πιν.4Β Παράμετροι Σεναρίου Εκτέλεσης (Rate/Power)

IoT Device	Type	Latency
Sensor	Deterministic	6ms
Actuator	Deterministic	1ms

Πιν.4Γ Παράμετροι Σεναρίου Εκτέλεσης (IoT Devices)

Η μέθοδος ***CreateFogDevice()*** στην κλάση Main παραμετροποιείται με τις τιμές των Πιν.4Α,4Β,4Γ. Για το σενάριο εκτέλεσης χρησιμοποιούνται τιμές με τις εξής παραδοχές.

Επεξεργαστική ισχύς CLOUD = 14* Edge

RAM CLOUD = 10* FoG = 40* Edge

Bandwidth CLOUD <-> FoG 10Mbit,

FoG <-> Edge 1Mbit

Επειδή ο κύριος ρόλος των FoG Devices μεσαίου επιπέδου είναι αυτός της λειτουργίας ως φορέα middleware υπηρεσίας (MainModule) και πυλών gateways περιορίσαμε τεχνητά την επεξεργαστική τους ισχύ (2800 MIPS) σε επίπεδο ελάχιστα μικρότερο των Edge devices για να εξετάσουμε καλύτερα την λειτουργία του δικτύου όσων αφορά την μεταφορά και εγκατάσταση client modules στα τελευταία.

Οι τιμές Uplink Bandwidth και Uplink Latency στην ***CreateFogDevice()*** του επιπέδου 0 έχουν συμπληρωθεί για λόγους λειτουργικούς με εικονικές τιμές 100, 1 για τη σωστή λειτουργία του iFogSim.

Επίσης οι καθορισμένες τιμές κόστους φαίνονται στο πίνακα 6.

Global	Cost per MeM	Cost Per Storage	Cost per Bw
3.0	0.05	0.001	0.0
Πιν.6 Κόστος (Double x values)			

Άρα στα πλαίσια της κατανομής πολιτικής πόρων έχουμε προτεραιότητα σε Κόστος Υλικών, Κόστος Χρήσης Μνήμης και Κόστος αποθήκευσης στο Cloud. Ορίστηκε μηδενικό κόστος χρήσης εύρους δικτύου καθώς είναι ιδιόκτητο.

Χαρακτηριστικές Τιμές Ροών Tuple και Modules

Module Type	Resource Requirement Parameters			
	MIPS	**RAM**	**Size**	**Bandwidth**
Client Module	10	1000	1000	100
Main Module	50	1500	4000	800
Storage Module	10	50	12000	100

Πιν.5 Παράμετροι Στοιχείων Ενοτήτων Εφαρμογής (Modules)

Για τα 3 τμήματα της αρθρωτής εφαρμογής (Client-Main-Storage) ορίσαμε διαφορετικές βαρύτητες.

Client Module : Μέσο μέγεθος σε απαιτήσεις δικτύου στην μνήμη, μικρές απαιτήσεις σε αποθηκευτικό χώρο, επεξεργαστική ισχύ και εύρος δίαυλου επικοινωνίας.

Main Module : Μέσο Μέγεθος σε απαιτήσεις δικτύου σε αποθήκευση και μνήμη , υψηλές απαιτήσεις σε επεξεργαστική ισχύ και εύρος δίαυλου επικοινωνίας.

Storage Module : Μικρό μέγεθος σε απαιτήσεις δικτύου στη μνήμη, σε επεξεργαστική ισχύ και εύρος δίαυλου επικοινωνίας, υψηλές απαιτήσεις σε αποθηκευτικό χώρο.

FOG RESOURCE REQUIREMENTS	CPU			MEMORY			STORAGE			NETWORK		
	Low	Medium	High	Low	Medium	High	Low	Medium	High	Low	Medium	High
Client Module	x				x		x			x		
Main Module			x		x			x				x
Storage Module	x			x					x	x		

Πιν.6 Βαρύτητες Απαιτήσεων σε Πόρους Υπολογιστικού Δικτύου Ομίχλης

DAG Edges	Tuple Direction	CPU LENGTH	NETWORK LENGHT
IoT Sensor→Client Module	TUPLE UP	100	200
Client Module→Main Module	TUPLE UP	6000	600
Main Module→Storage Module	TUPLE UP	1000	300
Main Module→Client Module	TUPLE DOWN	100	50
Client Module→ Actuator	TUPLE DOWN	100	50

Πιν.7 Παράμετροι Ροών Tuple

Κατά αντιστοιχία με τον πίνακα 6 της βαρύτητας σε πόρους υπολογιστικού δικτύου ομίχλης, παραμετροποιούνται και οι ροές tuple όπως βλέπουμε στο πίνακα 7.

Με την επιλογή των συγκεκριμένων παραμέτρων ουσιαστικά έχουμε κάνει μια εφαρμογή πολιτικής χρήσης πόρων μέσω παραμετροποίησης των ροών TUPLE και τον στοιχείων των τμημάτων της αρθρωτής εφαρμογής (Modules).

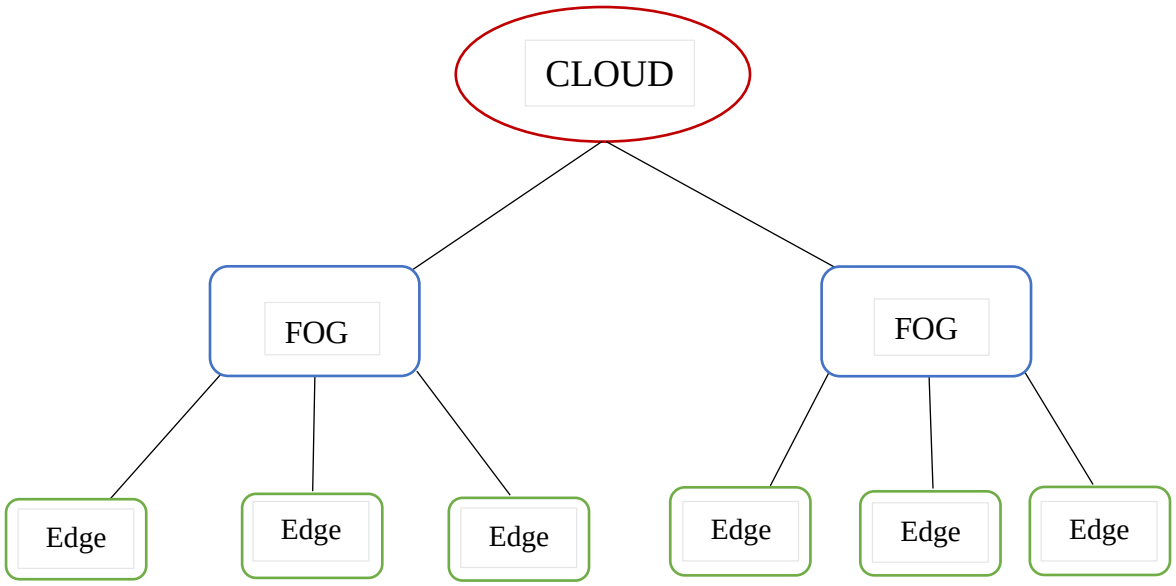
Η προσαρμογή αυτής της πολιτικής χρήσης πόρων πραγματοποιείται με το κατάλληλο προγραμματισμό στις κλάσεις Main, MyController, MyModule όπως αναλύθηκε στο 3^{ο} βήμα του Σχ.25.

4.3 ΣΕΝΑΡΙΑ ΥΛΟΠΟΙΗΣΗΣ

Σενάριο Εκτέλεσης 1

G2 E3

CLOUD - 2 Fog (Lvl1) - 6 Edge Devices (Lvl2)

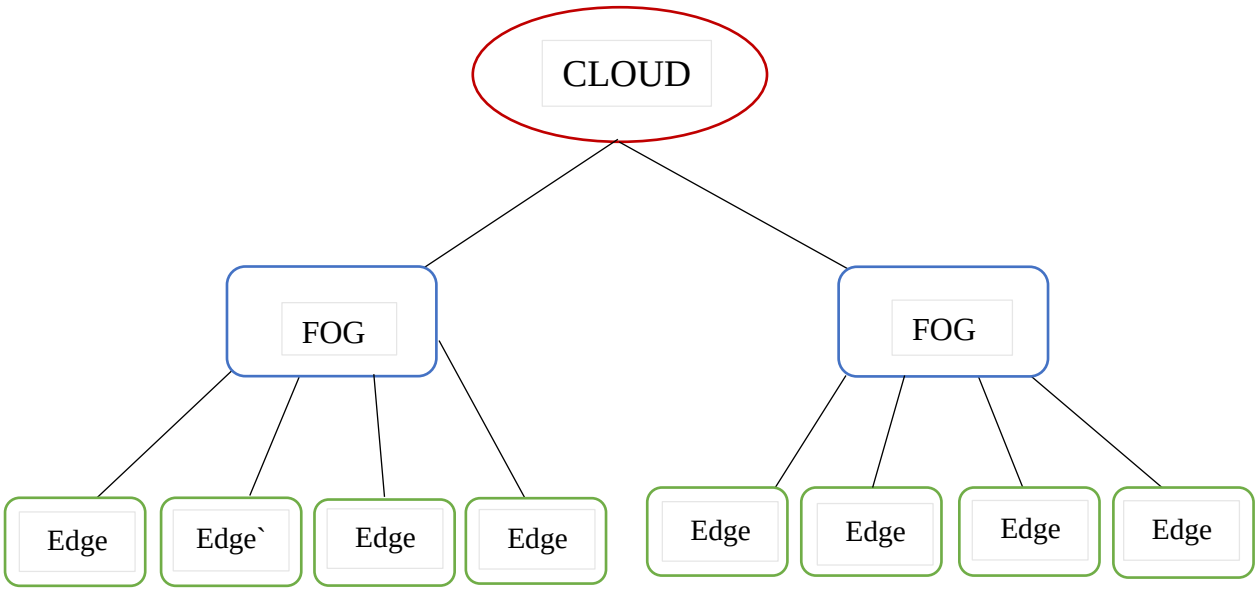


CLOUD	Level 0
FOG	Level 1
Edge	Level 2

Σενάριο Εκτέλεσης 2

G2 E4

CLOUD - 2 Fog (Lvl1) - 8 Edge Devices (Lvl2)

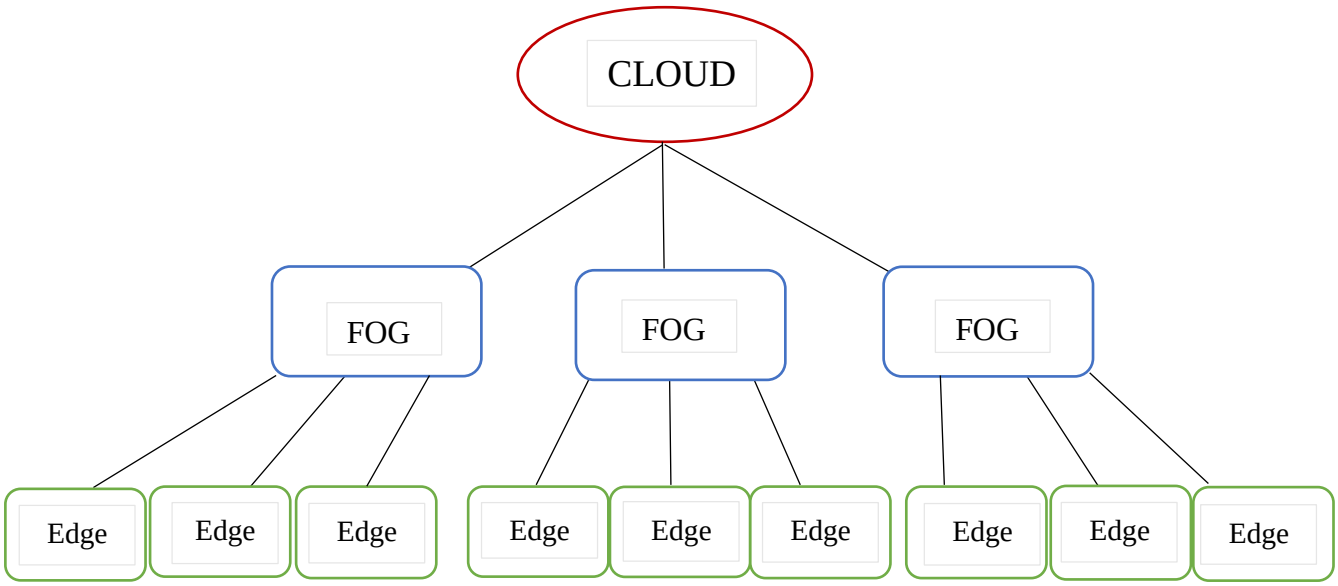


CLOUD	Level 0
FOG	Level 1
Edge	Level 2

Σενάριο Εκτέλεσης 3

G3 E3

CLOUD - 3 Fog (Lvl1) - 9 Edge Devices (Lvl2)

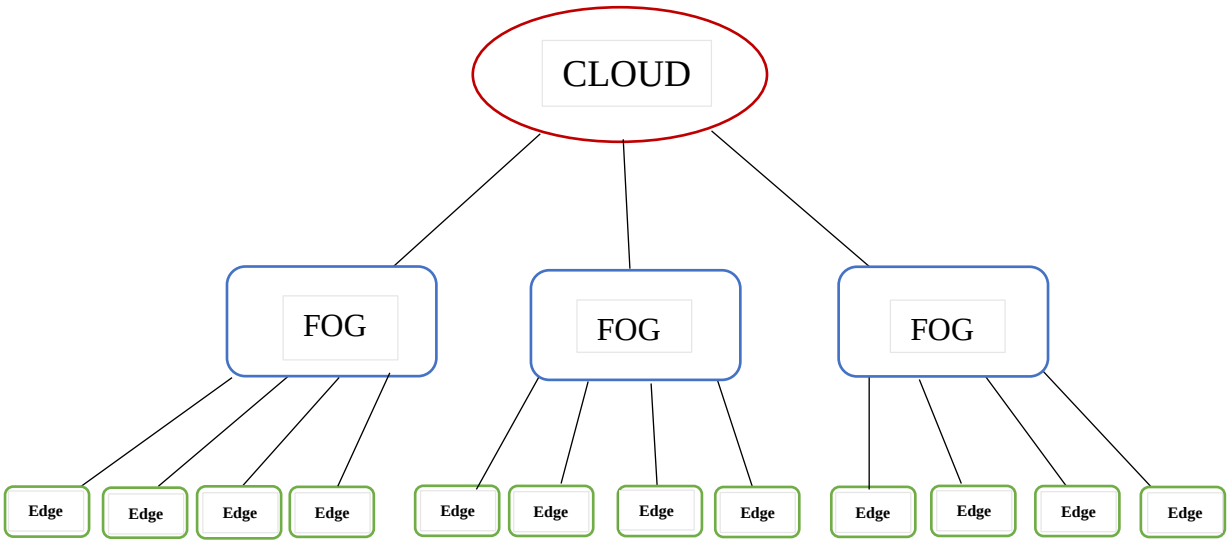


CLOUD	Level 0
FOG	Level 1
Edge	Level 2

Σενάριο Εκτέλεσης 4

G3 E4

CLOUD - 3 Fog (Lvl1) - 12 Edge Devices (Lvl2)



CLOUD	Level 0
FOG	Level 1
Edge	Level 2

5. ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΡΟΣΟΜΟΙΩΣΗΣ ΚΑΙ ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ

		Σενάρια Υλοποίησης				
		1	2	3	4	
		G2 E3	G2 E4	G3 E3	G3 E4	
cloud	ENERGY CONSUMPTION	431306,6372	429969,5843	447394,724	399705,8036	**CLOUD**
g0		31882,46926	27420,56	27639,95456	25029,99	
g1		29811,13	31880,97515	31880,2281	25029,99	**FOG**
g2				27420,56	31877,98694	
e-0-0		26206,24215	26210,32687	26214,02349	26210,32688	
e-0-1		26136,74456	26166,68012	26186,02213	26192,83	
e-0-2		25031,272	26192,83	26197,90091	26192,83	
e-0-3			26192,83		26192,83	
e-1-0		26192,83	25134,65717	26171,57925	26192,83	
e-1-1		26192,83	26189,66147	26197,92	26192,83	**EDGE**
e-1-2		26192,83	26192,83	25388,99902	26192,83	
e-1-3			26192,83		26192,83	
e-2-0				26197,92	24854,73263	
e-2-1				26197,92	26192,83	
e-2-2				26197,92	26192,83	
e-2-3					26192,83	
Cost of Execution Cloud		44951,18188	43055,61313	67759,60875	150	
Network Usage		2008,33333	2656	3789	2784	
Execution TIME		404	4006	32945	1500	

Η αρίθμηση των Nodes είναι 0,1,2 από αριστερά προς τα δεξιά
π.χ. g0 1ο FoG node αριστερά
e-2-3 4ο Edge node που συνδέεται με το 3ο FoG Node δεξιά

Πιν.8Α Αποτελέσματα Προσομοίωσης Σεναρίων Υλοποίησης *(Energy,Time,Cost)

TUPLE CPU EXECUTION DELAY			APPLICATION LOOP DELAYS	
Scenario 1				
G2 E3	RawData	80,09994	==========	
	Result Data	5,37147	[IoTSensor, clientModule, mainModule, clientModule, IoTActuator] --->	101,96979
	IoT Sensor	0,24470		
	Store Data	0,38460		
Scenario 2				
G2 E4	RawData	108,66539	==========	
	Result Data	1,67887	[IoTSensor, clientModule, mainModule, clientModule, IoTActuator] --->	93,03215
	IoT Sensor	0,25625		
	Store Data	0,41270		
Scenario 3				
G3 E3	RawData	4,81918	==========	
	Result Data	45,12714	[IoTSensor, clientModule, mainModule, clientModule, IoTActuator] --->	162,87390
	IoT Sensor	44,99998		
	Store Data	0,30204		
Scenario 4				
G4 E4	RawData		==========	
	Result Data		[IoTSensor, clientModule, mainModule, clientModule, IoTActuator] --->	Ø
	IoT Sensor	0,38125		
	Store Data			

Πιν.8Β Αποτελέσματα Προσομοίωσης Σεναρίων Υλοποίησης (Tuple Delays)

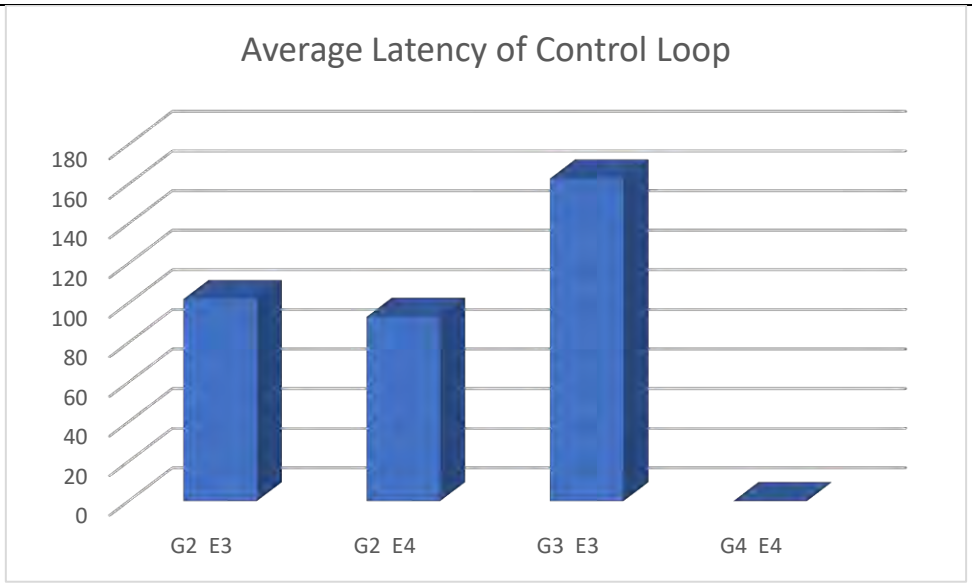
Παρατηρούμε ότι **στο σενάριο 4 της προσομοίωσης δεν ολοκληρώνεται ο κύκλος ελέγχου** (IoTSensor, clientModule, mainModule, clientModule, IoTActuator). Αυτό συμβαίνει διότι υπάρχει έλεγχος βρόχου στην πολιτική διαχείρισης (Σχ.31), ο οποίος ικανοποιείται πριν την ολοκλήρωση του διαμοιρασμού client modules στα Edge Devices και αφορά του διαθέσιμους πόρους στις FoG συσκευές.

Το σενάριο 4 όμως είναι ιδιαίτερα χρήσιμο για να εντοπιστούν τα όρια χωρητικότητας του δικτύου FoG σε edge devices (<12) για τις συγκεκριμένες παραμέτρους δικτύου (Πιν.4Α)

Προφανώς, αν ήταν επιθυμητή η επέκταση του δικτύου σε περισσότερες End Devices πρέπει να αυξηθεί η διαθεσιμότητα πόρων (MIPS,RAM) στις συσκευές FoG του επιπέδου 1.

5.1 ΔΙΑΓΡΑΜΜΑΤΑ - ΕΞΑΓΩΓΗ ΣΥΜΠΕΡΑΣΜΑΤΩΝ

ΜΕΣΗ ΚΑΘΥΣΤΕΡΗΣΗ ΚΥΚΛΟΥ ΕΛΕΓΧΟΥ



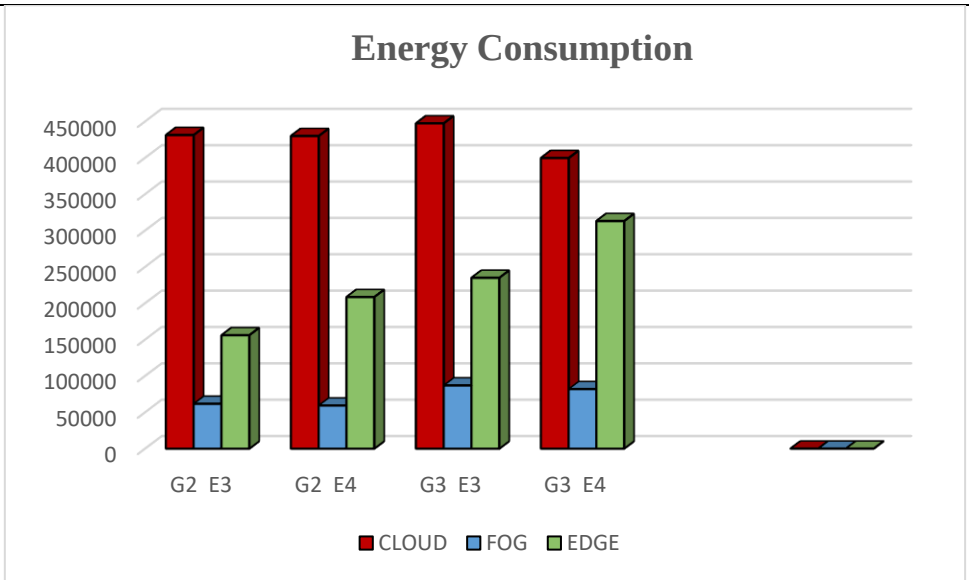
Σχ.32 Μέση Καθυστέρηση Κύκλου Ελέγχου Σεναρίων Υλοποίησης

Παρατηρείται έντονη συσχέτιση ανάμεσα στη μέση καθυστέρηση του κύκλου ελέγχου και τον αριθμό των FoG και Edge Devices.

Με παρατήρηση των επιμέρους καθυστερήσεων (average CPU delays, Πιν.8Β) , επέκταση του δικτύου προς τα κάτω για ίδιο αριθμό FoG devices στο επίπεδο 1 με περισσότερα Edge Devices στο επίπεδο 2,μειώνει τη συνολικό χρόνο ολοκλήρωσης κύκλου ελέγχου.

Αντιθέτως, αυξάνεται αρκετά ο χρόνος κύκλου ελέγχου σε κάθε προσθήκη FoG Device επιπέδου 1(Σενάριο 3).Γεγονός που είναι εύκολα κατανοητό αφού αυξάνεται λόγω της αρχιτεκτονικής αρκετά ο αριθμός των συνολικών hops των ροών tuple από το επίπεδο 2 στο επίπεδο 1.Στο σενάριο 4 όπως αναλύθηκε είχαμε διακοπή λόγω υπέρβασης ορίου διαθέσιμων πόρων.

ΚΑΤΑΝΑΛΩΣΗ ΕΝΕΡΓΕΙΑΣ ΣΤΟ ΔΙΚΤΥΟ



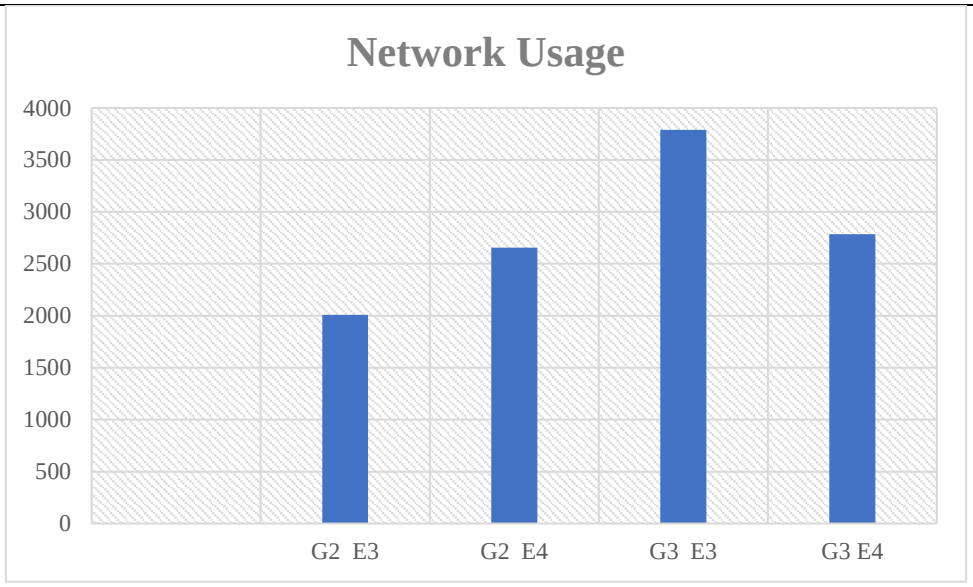
Σχ.33 Κατανάλωση Ενέργειας στο Δίκτυο

Είναι εύκολα κατανοητό (Σχ.33) ότι για τον ίδιο αριθμό FoG devices επιπέδου 1 αν αυξήσουμε τον αριθμό των Edge devices ανά κόμβο FoG έχουμε αύξηση της κατανάλωσης στις END devices με ελάχιστη πτώση στις αντίστοιχες συσκευές FoG.

Επομένως η πολιτική κατανομής πόρων που εφαρμόστηκε στο δίκτυο υπολογιστικής ομίχλης με την προσομοίωση που εκτελέστηκε στο iFogSim, λειτουργεί όπως ήταν αναμενόμενο. Δηλαδή, με χαμηλές δυνατότητες υπολογιστικής ισχύος στο μεσαίο επίπεδο έχουμε λειτουργία άψογη στο χαμηλότερο επίπεδο και μετακίνηση του όγκου της επεξεργασίας και διασπορά του φόρτου δικτύου προς τις χαμηλότερες συσκευές.

Σε όλα τα σενάρια η κατανάλωση ενέργειας στο Cloud παραμένει στα ίδια επίπεδα με μικρές διακυμάνσεις για περίπτωση Cloud only placement. Μάλιστα το άθροισμα της κατανάλωσης Fog και EDGE συσκευών σε κάθε σενάριο Edge-Ward τοποθέτησης είναι μικρότερο από την αντίστοιχη κατανάλωση ενέργειας σε Cloud Only τοποθέτηση, ακόμη και στην 4^{η} περίπτωση πού έχουμε 3 FoG και 12 EDGE Devices.

ΧΡΗΣΗ ΠΟΡΩΝ ΕΥΡΟΥΣ ΔΙΚΤΥΟΥ



Σχ.34 Χρήση Πόρων Εύρους Δικτύου

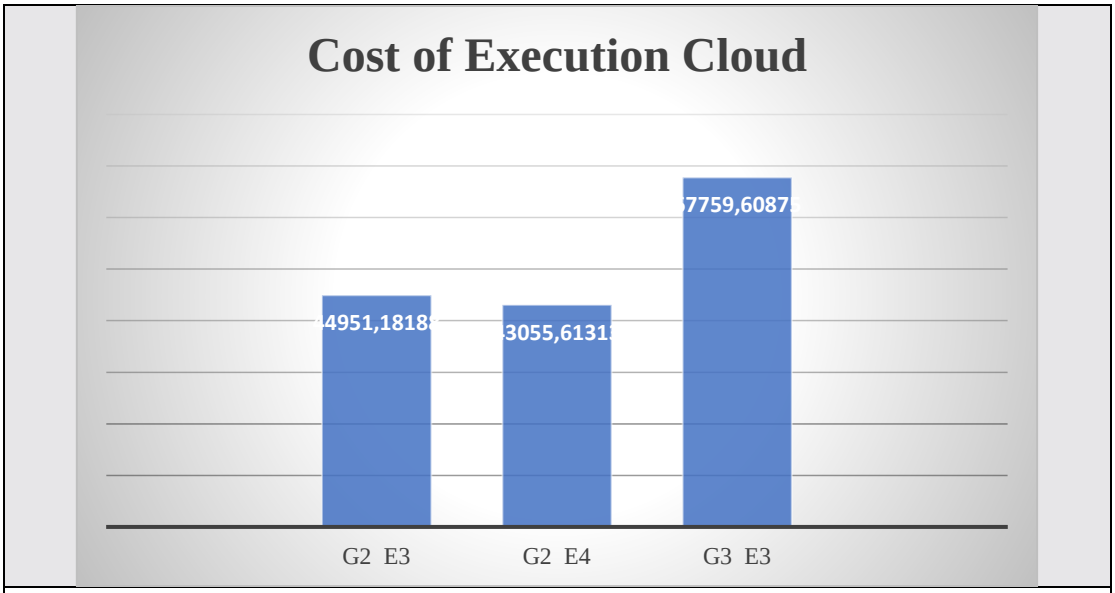
Αναμενόμενο διάγραμμα ομαλής αύξηση χρήσης εύρους δικτύου με την προσθήκη περισσότερων devices. Το 4ο σενάριο δε αντιπροσωπεύει πραγματική τιμή χρήσης δικτύου για σύγκριση καθώς ο βρόχος λειτουργίας σταματάει πριν ολοκληρωθεί ο διαμοιρασμός όλων των client modules και στις 12 συσκευές.

ΜΕΣΟΣ ΧΡΟΝΟΣ ΕΚΤΕΛΕΣΗΣ ΠΡΟΣΟΜΟΙΩΣΗΣ

Σχ.35 Μέσος Χρόνος Εκτέλεσης Προσομοίωσης

Εκθετική αύξηση του χρόνου της προσομοίωσης με την προσθήκη συσκευών EDGE αλλά κυρίως FoG. Κάτι το οποίο είναι αναμενόμενο λόγω της αρχιτεκτονικής του iFogSim, αφού με την προσθήκη συσκευών και επομένως και αισθητήρων και ενεργοποιητών, ο αριθμός των hops των tuple από το κάθε φύλο της δεντρικής δομής προς τη ρίζα και πάλι κάτω, αυξάνεται εκθετικά.

ΚΟΣΤΟΣ ΑΠΟΘΗΚΕΥΣΗΣ ΣΤΟ CLOUD



Σχ.36 Κόστος αποθήκευσης (storage data) στο Cloud

Αύξηση του κόστους αποθήκευσης με προσθήκη συσκευών κατά αναλογία με την κατανάλωση ενέργειας στο δίκτυο υπολογιστικής ομίχλης του σχήματος 23. Για ίδιο αριθμό Fog Devices με προσθήκη περισσοτέρων Edge στο κλάδο τους, μειώνεται λίγο το κόστος αποθήκευσης στο Cloud λόγω καλύτερη λειτουργίας της αρχιτεκτονικής των modules.

Κάθε μια προσθήκη FoG συσκευών του επιπέδου 1 αυξάνει σημαντικά το κόστος λόγω αύξησης των tuple hops και επι μέρους writes στο storage module που ξεκινούν από το κάθε αισθητήρα.

ΠΡΟΣΟΜΟΙΩΣΕΙΣ ΚΑΙ ΣΥΜΠΕΡΑΣΜΑΤΑ

Στα πλαίσια του μοντέλου τοποθέτησης της αρθρωτής εφαρμογής Main Module-Client Module-Storage Module που εφαρμόσαμε σε δίκτυο υπολογιστικής ομίχλης Βιομηχανίας 4.0 με βάση τα μοντέλα Εικονικοποίησης και PTPN που είδαμε στο προηγούμενο κεφάλαιο και ενσωματώνονται στον προσομοιωτή iFogSim όπως αναλύσαμε στο κεφάλαιο 2, έπειτα από την προσομοίωση που εκτελέστηκε στον προσομοιωτή iFogSim καταλήξαμε στα εξής συμπεράσματα.

Με προσθήκη στο υπολογιστικό δίκτυο ομίχλης συσκευών EDGE έχουμε βελτίωση της κατανομής ενέργειας ,μειώνοντας το βάρος επεξεργασίας στις συσκευές FoG του μεσαίου επιπέδου των οποίων οι πόροι μπορούν να χρησιμοποιηθούν και για άλλες χρήσεις παρόλη την μεγάλη αύξηση των αισθητήρων και ενεργοποιητών από όπου ξεκινούν και καταλήγουν οι ροές δεδομένων tuples στο κατώτερο επίπεδο.

Παράλληλα , έχουμε μείωση του κόστους αποθήκευσης στο CLOUD. Η αύξηση σε χρήση πόρων εύρους δικτύου και στην κατανάλωση ενέργειας είναι σε καλό ρυθμό και σε χαμηλά επίπεδα έναντι μιας καθαρής Cloud υλοποίησης για ίδιο αριθμό τερματικών συσκευών.

Πρόβλημα γενικά στη λειτουργία δικτύου εμφανίζεται μόνο όταν προστίθενται FoG συσκευές στο μεσαίο επίπεδο κατά την επέκταση του, εφόσον η διαθεσιμότητα των πόρων τους παραμένει σε χαμηλό επίπεδο σε σύγκριση με τις συσκευές EDGE του κατώτερου επιπέδου. Το γεγονός αυτό μπορεί να διερευνηθεί μελλοντικά μέσω βελτιστοποίησης της λειτουργίας του μεσαίου επιπέδου η προσθήκη και άλλου επιπέδου FoG συσκευών ανάμεσα στο 1^{ο} και 2^{ο} επίπεδο της προσομοίωσης.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Ashok Sutagundar, Sangeeta B Shahapur, ***Development of Fog based Dynamic Resource Allocation and Pricing Model in IoT***, 2018 Second International Conference on Green Computing and Internet of Things (ICGCIoT) ,16-18 Aug. 2018.

2. Rajkumar Buyya, Satish Narayana Srirama, ***Modeling and Simulation of Fog and Edge Computing Environments Using iFogSim Toolkit***, Wiley Telecom, Page(s): 433 – 465, 2019.

3. Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, Marimuthu Palaniswami, ***Internet of Things (IoT): A vision, architectural elements, and future directions***, Elsevier B.V, 2013./

4. Dimitrios Georgakopoulos , Prem Prakash Jayaraman , Maria Fazia , Massimo Villari , Rajiv Ranjan***, Internet of Things and Edge Cloud Computing Roadmap for Manufacturing***, IEEE Cloud Computing Volume: 3 , Issue: 4 , July-Aug. 2016.

5. Mohammad Aazam, Sherali Zeadally, Khaled A. Harras ***Deploying Fog Computing in Industrial Internet of Things and Industry 4.0***, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, VOL. 14, NO. 10, OCTOBER 2018.

6. Vangelis Gazis , Alessandro Leonardi , Kostas Mathioudakis , Konstantinos Sasloglou, Panayotis Kikiras AGT Group (R&D) GmbH, Germany, Raghuram Sudhaakar Cisco Systems Inc., USA ***Components of fog computing in an industrial internet of things context*** , 12th Annual IEEE International Conference on Sensing, Communication, and Networking - Workshops (SECON Workshops) ,2015.

7. Harshit Gupta , Amir Vahid Dastjerdi , Soumya K. Ghosh, Rajkumar Buyya, ***iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments*** , Wiley Telecom , 5 May 2017.

8. Jianhua Li, Jiong Jin , Dong Yuan, Hongke Zhang, ***Virtual Fog: A Virtualization Enabled Fog Computing Framework for Internet of Things*** , IEEE INTERNET OF THINGS JOURNAL, VOL. 5, NO. 1, FEBRUARY 2018.

9. Bonomi F, Milito R, Natarajan P, Zhu J. ***Fog computing: a platform for internet of things and analytics. In: Big Data and Internet of Things A Roadmap for Smart Environments*** Springer; 169-186, 2014.

10. Apostolos Galanopoulos , George Iosifidis, Theodoros Salonidis ***Optimizing Data Analytics in Energy Constrained IoT Networks*** , School of Computer Science and Statistics, Trinity College Dublin, Ireland , IBM T.J. Watson Research Center, Yorktown Heights, NY, USA

11. Lina Ni, Jinquan Zhang, Changjun Jiang, Chungang Yan, Kan Yu ***Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets***,IEEE INTERNET OF THINGS JOURNAL, VOL. 4, NO. 5, OCTOBER 2017.

ΠΙΝΑΚΕΣ

ΣΧΗΜΑΤΑ

ΠΑΡΑΡΤΗΜΑ Α - ΚΩΔΙΚΑΣ iFogSim

MAIN CLASS - TestApplication.java

```
package org.fog.test.perfeval;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Random;

import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.MyApplication;
import org.fog.application.selectivity.FractionalSelectivity;
import org.fog.entities.FogBroker;
import org.fog.entities.FogDeviceCharacteristics;
import org.fog.entities.MyActuator;
import org.fog.entities.MyFogDevice;
```

```
import org.fog.entities.MySensor;
import org.fog.entities.Tuple;
import org.fog.placement.ModuleMapping;
import org.fog.placement.MyController;
import org.fog.placement.MyModulePlacement;
import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.FogLinearPowerModel;
import org.fog.utils.FogUtils;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.DeterministicDistribution;
/**
 * Simulation setup for Fog Case Study 1 - 2 Clients(Smartphones) 1 IoT Sensor 1 Actuator(display) 1 Fog Device per Branch
 * @author Jim Lokas
 *
 */

public class TestApplication {
	static List<MyFogDevice> fogDevices = new ArrayList<MyFogDevice>();
	static Map<Integer,MyFogDevice> deviceById = new HashMap<Integer,MyFogDevice>();
	static List<MySensor> sensors = new ArrayList<MySensor>();
	static List<MyActuator> actuators = new ArrayList<MyActuator>();
	static List<Integer> idOfEndDevices = new ArrayList<Integer>();
	static Map<Integer, Map<String, Double>> deadlineInfo = new HashMap<Integer, Map<String, Double>>();
	static Map<Integer, Map<String, Integer>> additionalMipsInfo = new HashMap<Integer, Map<String, Integer>>();

	static boolean CLOUD = false;
```

```
static int numOfGateways = 2;
static int numOfEndDevPerGateway = 3;
static double sensingInterval = 5;

public static void main(String[] args) {

	Log.printLine("Starting TestApplication...");

	try {
		Log.disable();
		int num_user = 1;
		Calendar calendar = Calendar.getInstance();
		boolean trace_flag = false;
		CloudSim.init(num_user, calendar, trace_flag);
		String appId = "test_app";
		FogBroker broker = new FogBroker("broker");

		createFogDevices(broker.getId(), appId);

		MyApplication application = createApplication(appId,
broker.getId());
		application.setUserId(broker.getId());

		ModuleMapping moduleMapping =
ModuleMapping.createModuleMapping();

		moduleMapping.addModuleToDevice("storageModule",
"cloud");
		for(int i=0;i<idOfEndDevices.size();i++)
		{
			MyFogDevice fogDevice =
deviceById.get(idOfEndDevices.get(i));
```

```
                    moduleMapping.addModuleToDevice("clientModule",
fogDevice.getName());
                }



                MyController controller = new MyController("master-
controller", fogDevices, sensors, actuators);


                controller.submitApplication(application, 0, new
MyModulePlacement(fogDevices, sensors, actuators, application,
moduleMapping,"mainModule"));



        TimeKeeper.getInstance().setSimulationStartTime(Calendar.getInstance().get
TimeInMillis());


                CloudSim.startSimulation();


                CloudSim.stopSimulation();


                Log.printLine("TestApplication finished!");
            } catch (Exception e) {
                e.printStackTrace();
                Log.printLine("Unwanted errors happen");
            }
        }


        private static double getvalue(double min, double max)
        {
            Random r = new Random();
            double randomValue = min + (max - min) * r.nextDouble();
            return randomValue;
```

```
	}

	private static int getvalue(int min, int max)
	{
		Random r = new Random();
		int randomValue = min + r.nextInt()%(max - min);
		return randomValue;
	}

	private static void createFogDevices(int userId, String appId) {
		MyFogDevice cloud = createFogDevice("cloud", 44800, 40000, 100,
10000, 0, 0.01, 16*103, 16*83.25);
		cloud.setParentId(-1);
		fogDevices.add(cloud);
		deviceById.put(cloud.getId(), cloud);

		for(int i=0;i<numOfGateways;i++){
			addGw(i+"", userId, appId, cloud.getId());
		}

	}

	private static void addGw(String gwPartialName, int userId, String appId, int
parentId){
		MyFogDevice gw = createFogDevice("g-"+gwPartialName, 2800,
4000, 10000, 10000, 1, 0.0, 107.339, 83.4333);
		fogDevices.add(gw);
		deviceById.put(gw.getId(), gw);
		gw.setParentId(parentId);
		gw.setUplinkLatency(4);
		for(int i=0;i<numOfEndDevPerGateway;i++){
```

```
                    String endPartialName = gwPartialName+"-"+i;
                    MyFogDevice end  = addEnd(endPartialName, userId, appId,
gw.getId());
                    end.setUplinkLatency(2);
                    fogDevices.add(end);
                    deviceById.put(end.getId(), end);
                }

        }

        private static MyFogDevice addEnd(String endPartialName, int userId, String
appId, int parentId){
                MyFogDevice end = createFogDevice("e-"+endPartialName, 3200,
1000, 10000, 270, 2, 0, 87.53, 82.44);
                end.setParentId(parentId);
                idOfEndDevices.add(end.getId());
                MySensor sensor = new MySensor("s-"+endPartialName,
"IoTSensor", userId, appId, new DeterministicDistribution(sensingInterval)); // inter-
transmission time of EEG sensor follows a deterministic distribution
                sensors.add(sensor);
                MyActuator actuator = new MyActuator("a-"+endPartialName, userId,
appId, "IoTActuator");
                actuators.add(actuator);
                sensor.setGatewayDeviceId(end.getId());
                sensor.setLatency(6.0);  // latency of connection between EEG sensors
and the parent Smartphone is 6 ms
                actuator.setGatewayDeviceId(end.getId());
                actuator.setLatency(1.0);  // latency of connection between Display
actuator and the parent Smartphone is 1 ms
                return end;
        }

        private static MyFogDevice createFogDevice(String nodeName, long mips,
```

```
                int ram, long upBw, long downBw, int level, double
ratePerMips, double busyPower, double idlePower) {
        List<Pe> peList = new ArrayList<Pe>();
        peList.add(new Pe(0, new PeProvisionerOverbooking(mips)));
        int hostId = FogUtils.generateEntityId();
        long storage = 1000000;
        int bw = 10000;

        PowerHost host = new PowerHost(
                hostId,
                new RamProvisionerSimple(ram),
                new BwProvisionerOverbooking(bw),
                storage,
                peList,
                new StreamOperatorScheduler(peList),
                new FogLinearPowerModel(busyPower, idlePower)
            );
        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);
        String arch = "x86";
        String os = "Linux";
        String vmm = "Xen";
        double time_zone = 10.0;
        double cost = 3.0;
        double costPerMem = 0.05;
        double costPerStorage = 0.001;
        double costPerBw = 0.0;
        LinkedList<Storage> storageList = new LinkedList<Storage>();
        FogDeviceCharacteristics characteristics = new
FogDeviceCharacteristics(
                arch, os, vmm, host, time_zone, cost, costPerMem,
```

```
                costPerStorage, costPerBw);

        MyFogDevice fogdevice = null;
        try {
            fogdevice = new MyFogDevice(nodeName, characteristics,
                    new AppModuleAllocationPolicy(hostList),
storageList, 10, upBw, downBw, 0, ratePerMips);
        } catch (Exception e) {
            e.printStackTrace();
        }

        fogdevice.setLevel(level);
        fogdevice.setMips((int) mips);
        return fogdevice;
    }

    @SuppressWarnings({"serial" })
    private static MyApplication createApplication(String appId, int userId){

        MyApplication application = MyApplication.createApplication(appId,
userId);
        application.addAppModule("clientModule",10, 1000, 1000, 100);
        application.addAppModule("mainModule", 50, 1500, 4000, 800);
        application.addAppModule("storageModule", 10, 50, 12000, 100);

        application.addAppEdge("IoTSensor", "clientModule", 100, 200,
"IoTSensor", Tuple.UP, AppEdge.SENSOR);
        application.addAppEdge("clientModule", "mainModule", 6000, 600  ,
"RawData", Tuple.UP, AppEdge.MODULE);
        application.addAppEdge("mainModule", "storageModule", 1000, 300,
"StoreData", Tuple.UP, AppEdge.MODULE);
```

```
application.addAppEdge("mainModule", "clientModule", 100, 50,
"ResultData", Tuple.DOWN, AppEdge.MODULE);

application.addAppEdge("clientModule", "IoTActuator", 100, 50,
"Response", Tuple.DOWN, AppEdge.ACTUATOR);


application.addTupleMapping("clientModule", "IoTSensor",
"RawData", new FractionalSelectivity(1.0));

application.addTupleMapping("mainModule", "RawData",
"ResultData", new FractionalSelectivity(1.0));

application.addTupleMapping("mainModule", "RawData",
"StoreData", new FractionalSelectivity(1.0));

application.addTupleMapping("clientModule", "ResultData",
"Response", new FractionalSelectivity(1.0));



for(int id:idOfEndDevices)

{

Map<String,Double>moduleDeadline = new
HashMap<String,Double>();

moduleDeadline.put("mainModule", getvalue(3.00, 5.00));

Map<String,Integer>moduleAddMips = new
HashMap<String,Integer>();

moduleAddMips.put("mainModule", getvalue(0, 500));

deadlineInfo.put(id, moduleDeadline);

additionalMipsInfo.put(id,moduleAddMips);


}


final AppLoop loop1 = new AppLoop(new
ArrayList<String>(){{add("IoTSensor");add("clientModule");add("mainModule");ad
d("clientModule");add("IoTActuator");}});

List<AppLoop> loops = new ArrayList<AppLoop>(){{add(loop1);}};

application.setLoops(loops);

application.setDeadlineInfo(deadlineInfo);
```

```
                application.setAdditionalMipsInfo(additionalMipsInfo);

                return application;
        }
}
```

MyApplication.java

```
package org.fog.application;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.apache.commons.math3.util.Pair;
import org.cloudbus.cloudsim.UtilizationModelFull;
import org.fog.application.selectivity.SelectivityModel;
import org.fog.entities.Tuple;
import org.fog.scheduler.TupleScheduler;
import org.fog.utils.FogUtils;
import org.fog.utils.GeoCoverage;

/**
 * Class represents an application in the Distributed Dataflow Model.
 * @author Harshit Gupta
 *
 */
public class MyApplication {

	private String appId;
	private int userId;
	private GeoCoverage geoCoverage;
```

```
/**
 * List of application modules in the application
 */
private List<AppModule> modules;

/**
 * List of application edges in the application
 */
private List<AppEdge> edges;

/**
 * List of application loops to monitor for delay
 */
private List<AppLoop> loops;

private Map<String, AppEdge> edgeMap;

/**
 * Creates a plain vanilla application with no modules and edges.
 * @param appId
 * @param userId
 * @return
 */
public static MyApplication createApplication(String appId, int userId){
        return new MyApplication(appId, userId);
}

/**
 * Adds an application module to the application.
 * @param moduleName
```

```
     * @param ram
     */

    /**
     * Adds a non-periodic edge to the application model.
     * @param source
     * @param destination
     * @param tupleCpuLength
     * @param tupleNwLength
     * @param tupleType
     * @param direction
     * @param edgeType
     */
    public void addAppEdge(String source, String destination, double tupleCpuLength,
                           double tupleNwLength, String tupleType, int direction, int edgeType){
        AppEdge edge = new AppEdge(source, destination, tupleCpuLength, tupleNwLength, tupleType, direction, edgeType);
        getEdges().add(edge);
        getEdgeMap().put(edge.getTupleType(), edge);
    }

    /**
     * Adds a periodic edge to the application model.
     * @param source
     * @param destination
     * @param tupleCpuLength
     * @param tupleNwLength
     * @param tupleType
     * @param direction
```

```
     * @param edgeType
     */
    public void addAppEdge(String source, String destination, double periodicity, double tupleCpuLength,
                           double tupleNwLength, String tupleType, int direction, int edgeType){
        AppEdge edge = new AppEdge(source, destination, periodicity, tupleCpuLength, tupleNwLength, tupleType, direction, edgeType);
        getEdges().add(edge);
        getEdgeMap().put(edge.getTupleType(), edge);
    }

    /**
     * Define the input-output relationship of an application module for a given input tuple type.
     * @param moduleName Name of the module
     * @param inputTupleType Type of tuples carried by the incoming edge
     * @param outputTupleType Type of tuples carried by the output edge
     * @param selectivityModel Selectivity model governing the relation between the incoming and outgoing edge
     */
    public void addTupleMapping(String moduleName, String inputTupleType, String outputTupleType, SelectivityModel selectivityModel){
        AppModule module = getModuleByName(moduleName);
        module.getSelectivityMap().put(new Pair<String, String>(inputTupleType, outputTupleType), selectivityModel);
    }

    /**
     * Get a list of all periodic edges in the application.
     * @param srcModule
     * @return
     */
```

```
	public List<AppEdge> getPeriodicEdges(String srcModule){
		List<AppEdge> result = new ArrayList<AppEdge>();
		for(AppEdge edge : edges){
			if(edge.isPeriodic() && edge.getSource().equals(srcModule))
				result.add(edge);
		}
		return result;
	}

	public MyApplication(String appId, int userId) {
		setAppId(appId);
		setUserId(userId);
		setModules(new ArrayList<AppModule>());
		setEdges(new ArrayList<AppEdge>());
		setGeoCoverage(null);
		setLoops(new ArrayList<AppLoop>());
		setEdgeMap(new HashMap<String, AppEdge>());
	}

	public MyApplication(String appId, List<AppModule> modules,
			List<AppEdge> edges, List<AppLoop> loops, GeoCoverage
geoCoverage) {
		setAppId(appId);
		setModules(modules);
		setEdges(edges);
		setGeoCoverage(geoCoverage);
		setLoops(loops);
		setEdgeMap(new HashMap<String, AppEdge>());
		for(AppEdge edge : edges){
			getEdgeMap().put(edge.getTupleType(), edge);
		}
```

```
	}

	/**
	 * Search and return an application module by its module name
	 * @param name the module name to be returned
	 * @return
	 */
	public AppModule getModuleByName(String name){
		for(AppModule module : modules){
			if(module.getName().equals(name))
				return module;
		}
		return null;
	}

	/**
	 * Get the tuples generated upon execution of incoming tuple <i>inputTuple</i> by module named <i>moduleName</i>
	 * @param moduleName name of the module performing execution of incoming tuple and emitting resultant tuples
	 * @param inputTuple incoming tuple, whose execution creates resultant tuples
	 * @param sourceDeviceId
	 * @return
	 */
	public List<Tuple> getResultantTuples(String moduleName, Tuple inputTuple, int sourceDeviceId, int sourceModuleId){
		List<Tuple> tuples = new ArrayList<Tuple>();
		AppModule module = getModuleByName(moduleName);
		for(AppEdge edge : getEdges()){
			if(edge.getSource().equals(moduleName)){
```

```
                    Pair<String, String> pair = new Pair<String,
String>(inputTuple.getTupleType(), edge.getTupleType());


                    if(module.getSelectivityMap().get(pair)==null)
                        continue;
                    SelectivityModel selectivityModel =
module.getSelectivityMap().get(pair);
                    if(selectivityModel.canSelect()){
                        //TODO check if the edge is ACTUATOR, then
create multiple tuples
                        if(edge.getEdgeType() ==
AppEdge.ACTUATOR){
                            //for(Integer actuatorId :
module.getActuatorSubscriptions().get(edge.getTupleType())){
                                Tuple tuple = new Tuple(appId,
FogUtils.generateTupleId(), edge.getDirection(),
                                        (long)
(edge.getTupleCpuLength()),
        inputTuple.getNumberOfPes(),
                                        (long)
(edge.getTupleNwLength()),
        inputTuple.getCloudletOutputSize(),
        inputTuple.getUtilizationModelCpu(),
        inputTuple.getUtilizationModelRam(),
        inputTuple.getUtilizationModelBw()
                                        );
        tuple.setActualTupleId(inputTuple.getActualTupleId());
        tuple.setUserId(inputTuple.getUserId());
```

```
tuple.setAppId(inputTuple.getAppId());

tuple.setDestModuleName(edge.getDestination());

tuple.setSrcModuleName(edge.getSource());

tuple.setDirection(Tuple.ACTUATOR);

tuple.setTupleType(edge.getTupleType());

tuple.setSourceDeviceId(sourceDeviceId);

tuple.setSourceModuleId(sourceModuleId);
                                        //tuple.setActuatorId(actuatorId);


                                        tuples.add(tuple);
                                //}
                            }else{
                                Tuple tuple = new Tuple(appId,
FogUtils.generateTupleId(), edge.getDirection(),
                                        (long)
(edge.getTupleCpuLength()),

        inputTuple.getNumberOfPes(),
                                        (long)
(edge.getTupleNwLength()),

        inputTuple.getCloudletOutputSize(),

        inputTuple.getUtilizationModelCpu(),

        inputTuple.getUtilizationModelRam(),

        inputTuple.getUtilizationModelBw()
                                        );
```

```
tuple.setActualTupleId(inputTuple.getActualTupleId());
                                        tuple.setUserId(inputTuple.getUserId());
                                        tuple.setAppId(inputTuple.getAppId());

tuple.setDestModuleName(edge.getDestination());

tuple.setSrcModuleName(edge.getSource());
                                        tuple.setDirection(edge.getDirection());

tuple.setTupleType(edge.getTupleType());

tuple.setSourceModuleId(sourceModuleId);

                                        tuples.add(tuple);
                                }
                        }
                }
        }
        return tuples;
}

/**
 * Create a tuple for a given application edge
 * @param edge
 * @param sourceDeviceId
 * @return
 */
public Tuple createTuple(AppEdge edge, int sourceDeviceId, int
sourceModuleId){
        AppModule module = getModuleByName(edge.getSource());
        if(edge.getEdgeType() == AppEdge.ACTUATOR){
```

```
                for(Integer actuatorId :
module.getActuatorSubscriptions().get(edge.getTupleType())){
                    Tuple tuple = new Tuple(appId,
FogUtils.generateTupleId(), edge.getDirection(),
                            (long) (edge.getTupleCpuLength()),
                            1,
                            (long) (edge.getTupleNwLength()),
                            100,
                            new UtilizationModelFull(),
                            new UtilizationModelFull(),
                            new UtilizationModelFull()
                            );
                    tuple.setUserId(getUserId());
                    tuple.setAppId(getAppId());
                    tuple.setDestModuleName(edge.getDestination());
                    tuple.setSrcModuleName(edge.getSource());
                    tuple.setDirection(Tuple.ACTUATOR);
                    tuple.setTupleType(edge.getTupleType());
                    tuple.setSourceDeviceId(sourceDeviceId);
                    tuple.setActuatorId(actuatorId);
                    tuple.setSourceModuleId(sourceModuleId);

                    return tuple;
                }
            }else{
                Tuple tuple = new Tuple(appId, FogUtils.generateTupleId(),
edge.getDirection(),
                        (long) (edge.getTupleCpuLength()),
                        1,
                        (long) (edge.getTupleNwLength()),
                        100,
```

```
                                        new UtilizationModelFull(),
                                        new UtilizationModelFull(),
                                        new UtilizationModelFull()
                                        );
                        //tuple.setActualTupleId(inputTuple.getActualTupleId());
                        tuple.setUserId(getUserId());
                        tuple.setAppId(getAppId());
                        tuple.setDestModuleName(edge.getDestination());
                        tuple.setSrcModuleName(edge.getSource());
                        tuple.setDirection(edge.getDirection());
                        tuple.setTupleType(edge.getTupleType());
                        tuple.setSourceModuleId(sourceModuleId);

                        return tuple;
                }
                return null;
        }

        public String getAppId() {
                return appId;
        }
        public void setAppId(String appId) {
                this.appId = appId;
        }
        public List<AppModule> getModules() {
                return modules;
        }
        public void setModules(List<AppModule> modules) {
                this.modules = modules;
        }
```

```
public List<AppEdge> getEdges() {
        return edges;
}
public void setEdges(List<AppEdge> edges) {
        this.edges = edges;
}
public GeoCoverage getGeoCoverage() {
        return geoCoverage;
}
public void setGeoCoverage(GeoCoverage geoCoverage) {
        this.geoCoverage = geoCoverage;
}

public List<AppLoop> getLoops() {
        return loops;
}

public void setLoops(List<AppLoop> loops) {
        this.loops = loops;
}

public int getUserId() {
        return userId;
}

public void setUserId(int userId) {
        this.userId = userId;
}

public Map<String, AppEdge> getEdgeMap() {
```

```
		return edgeMap;
	}

	public void setEdgeMap(Map<String, AppEdge> edgeMap) {
		this.edgeMap = edgeMap;
	}

	////////////////////////////////////////////////////////////// inclusion
	private Map<Integer, Map<String, Double>> deadlineInfo;
	private Map<Integer, Map<String, Integer>> additionalMipsInfo;

	public Map<Integer, Map<String, Integer>> getAdditionalMipsInfo() {
		return additionalMipsInfo;
	}

	public void setAdditionalMipsInfo(
			Map<Integer, Map<String, Integer>> additionalMipsInfo) {
		this.additionalMipsInfo = additionalMipsInfo;
	}

	public void setDeadlineInfo(Map<Integer, Map<String, Double>>
deadlineInfo) {
		this.deadlineInfo = deadlineInfo;
	}

	public Map<Integer, Map<String, Double>> getDeadlineInfo() {
		return deadlineInfo;
	}

	public void addAppModule(String moduleName,int ram, int mips, long size,
long bw){
```

```
		String vmm = "Xen";

		AppModule module = new AppModule(FogUtils.generateEntityId(),
moduleName, appId, userId,

				mips, ram, bw, size, vmm, new TupleScheduler(mips,
1), new HashMap<Pair<String, String>, SelectivityModel>());


		getModules().add(module);


	}


}
```

MyFogDevice.java

```
package org.fog.entities;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Queue;

import org.apache.commons.math3.util.Pair;
import org.cloudbus.cloudsim.Cloudlet;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Pe;
import org.cloudbus.cloudsim.Storage;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.VmAllocationPolicy;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.CloudSimTags;
import org.cloudbus.cloudsim.core.SimEvent;
import org.cloudbus.cloudsim.power.PowerDatacenter;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.power.models.PowerModel;
import org.cloudbus.cloudsim.provisioners.RamProvisionerSimple;
import org.cloudbus.cloudsim.sdn.overbooking.BwProvisionerOverbooking;
import org.cloudbus.cloudsim.sdn.overbooking.PeProvisionerOverbooking;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
```

```
import org.fog.application.AppModule;
import org.fog.application.MyApplication;

import org.fog.policy.AppModuleAllocationPolicy;
import org.fog.scheduler.StreamOperatorScheduler;
import org.fog.utils.Config;
import org.fog.utils.FogEvents;
import org.fog.utils.FogUtils;
import org.fog.utils.Logger;
import org.fog.utils.ModuleLaunchConfig;
import org.fog.utils.NetworkUsageMonitor;
import org.fog.utils.TimeKeeper;

public class MyFogDevice extends PowerDatacenter {
	protected Queue<Tuple> northTupleQueue;
	protected Queue<Pair<Tuple, Integer>> southTupleQueue;

	protected List<String> activeMyApplications;

	protected Map<String, MyApplication> applicationMap;
	public Map<String, List<String>> appToModulesMap;
	protected Map<Integer, Double> childToLatencyMap;


	protected Map<Integer, Integer> cloudTrafficMap;

	protected double lockTime;

	/**
	 * ID of the parent Fog Device
```

```
 */
protected int parentId;

/**
 * ID of the Controller
 */
protected int controllerId;
/**
 * IDs of the children Fog devices
 */
protected List<Integer> childrenIds;

protected Map<Integer, List<String>> childToOperatorsMap;

/**
 * Flag denoting whether the link southwards from this FogDevice is busy
 */
protected boolean isSouthLinkBusy;

/**
 * Flag denoting whether the link northwards from this FogDevice is busy
 */
protected boolean isNorthLinkBusy;

protected double uplinkBandwidth;
protected double downlinkBandwidth;
protected double uplinkLatency;
protected List<Pair<Integer, Double>> associatedMyActuatorIds;

protected double energyConsumption;
```

```
	protected double lastUtilizationUpdateTime;
	protected double lastUtilization;
	private int level;

	protected double ratePerMips;

	protected double totalCost;

	protected Map<String, Map<String, Integer>> moduleInstanceCount;

	public MyFogDevice(
			String name,
			FogDeviceCharacteristics characteristics,
			VmAllocationPolicy vmAllocationPolicy,
			List<Storage> storageList,
			double schedulingInterval,
			double uplinkBandwidth, double downlinkBandwidth, double
uplinkLatency, double ratePerMips) throws Exception {
		super(name, characteristics, vmAllocationPolicy, storageList,
schedulingInterval);
		setCharacteristics(characteristics);
		setVmAllocationPolicy(vmAllocationPolicy);
		setLastProcessTime(0.0);
		setStorageList(storageList);
		setVmList(new ArrayList<Vm>());
		setSchedulingInterval(schedulingInterval);
		setUplinkBandwidth(uplinkBandwidth);
		setDownlinkBandwidth(downlinkBandwidth);
		setUplinkLatency(uplinkLatency);
		setRatePerMips(ratePerMips);
```

```
setAssociatedMyActuatorIds(new ArrayList<Pair<Integer,
Double>>());
for (Host host : getCharacteristics().getHostList()) {
host.setDatacenter(this);
}
setActiveMyApplications(new ArrayList<String>());
// If this resource doesn't have any PEs then no useful at all
if (getCharacteristics().getNumberOfPes() == 0) {
throw new Exception(super.getName()
+ " : Error - this entity has no PEs. Therefore,
can't process any Cloudlets.");
}
// stores id of this class
getCharacteristics().setId(super.getId());

applicationMap = new HashMap<String, MyApplication>();
appToModulesMap = new HashMap<String, List<String>>();
northTupleQueue = new LinkedList<Tuple>();
southTupleQueue = new LinkedList<Pair<Tuple, Integer>>();
setNorthLinkBusy(false);
setSouthLinkBusy(false);



setChildrenIds(new ArrayList<Integer>());
setChildToOperatorsMap(new HashMap<Integer, List<String>>());

this.cloudTrafficMap = new HashMap<Integer, Integer>();

this.lockTime = 0;

this.energyConsumption = 0;
```

```
		this.lastUtilization = 0;
		setTotalCost(0);
		setModuleInstanceCount(new HashMap<String, Map<String,
Integer>>());
		setChildToLatencyMap(new HashMap<Integer, Double>());
	}

	public MyFogDevice(
			String name, long mips, int ram,
			double uplinkBandwidth, double downlinkBandwidth, double
ratePerMips, PowerModel powerModel) throws Exception {
		super(name, null, null, new LinkedList<Storage>(), 0);

		List<Pe> peList = new ArrayList<Pe>();

		// 3. Create PEs and add these into a list.
		peList.add(new Pe(0, new PeProvisionerOverbooking(mips))); // need
to store Pe id and MIPS Rating

		int hostId = FogUtils.generateEntityId();
		long storage = 1000000; // host storage
		int bw = 10000;

		PowerHost host = new PowerHost(
				hostId,
				new RamProvisionerSimple(ram),
				new BwProvisionerOverbooking(bw),
				storage,
				peList,
				new StreamOperatorScheduler(peList),
				powerModel
```

```
                );

        List<Host> hostList = new ArrayList<Host>();
        hostList.add(host);

        setVmAllocationPolicy(new AppModuleAllocationPolicy(hostList));

        String arch = Config.FOG_DEVICE_ARCH;
        String os = Config.FOG_DEVICE_OS;
        String vmm = Config.FOG_DEVICE_VMM;
        double time_zone = Config.FOG_DEVICE_TIMEZONE;
        double cost = Config.FOG_DEVICE_COST;
        double costPerMem =
Config.FOG_DEVICE_COST_PER_MEMORY;
        double costPerStorage =
Config.FOG_DEVICE_COST_PER_STORAGE;
        double costPerBw = Config.FOG_DEVICE_COST_PER_BW;

        FogDeviceCharacteristics characteristics = new
FogDeviceCharacteristics(
                        arch, os, vmm, host, time_zone, cost, costPerMem,
                        costPerStorage, costPerBw);

        setCharacteristics(characteristics);

        setLastProcessTime(0.0);
        setVmList(new ArrayList<Vm>());
        setUplinkBandwidth(uplinkBandwidth);
        setDownlinkBandwidth(downlinkBandwidth);
        setUplinkLatency(uplinkLatency);
        setAssociatedMyActuatorIds(new ArrayList<Pair<Integer,
Double>>());
```

```
		for (Host host1 : getCharacteristics().getHostList()) {
			host1.setDatacenter(this);
		}
		setActiveMyApplications(new ArrayList<String>());
		if (getCharacteristics().getNumberOfPes() == 0) {
			throw new Exception(super.getName()
					+ " : Error - this entity has no PEs. Therefore,
can't process any Cloudlets.");
		}


		getCharacteristics().setId(super.getId());

		applicationMap = new HashMap<String, MyApplication>();
		appToModulesMap = new HashMap<String, List<String>>();
		northTupleQueue = new LinkedList<Tuple>();
		southTupleQueue = new LinkedList<Pair<Tuple, Integer>>();
		setNorthLinkBusy(false);
		setSouthLinkBusy(false);


		setChildrenIds(new ArrayList<Integer>());
		setChildToOperatorsMap(new HashMap<Integer, List<String>>());

		this.cloudTrafficMap = new HashMap<Integer, Integer>();

		this.lockTime = 0;

		this.energyConsumption = 0;
		this.lastUtilization = 0;
		setTotalCost(0);
```

```
                setChildToLatencyMap(new HashMap<Integer, Double>());
                setModuleInstanceCount(new HashMap<String, Map<String,
Integer>>());
        }

        /**
         * Overrides this method when making a new and different type of resource.
<br>
         * <b>NOTE:</b> You do not need to override {@link #body()} method, if
you use this method.
         *
         * @pre $none
         * @post $none
         */
        protected void registerOtherEntity() {

        }

        @Override
        protected void processOtherEvent(SimEvent ev) {
                switch(ev.getTag()){
                case FogEvents.TUPLE_ARRIVAL:
                        processTupleArrival(ev);
                        break;
                case FogEvents.LAUNCH_MODULE:
                        processModuleArrival(ev);
                        break;
                case FogEvents.RELEASE_OPERATOR:
                        processOperatorRelease(ev);
                        break;
                case FogEvents.SENSOR_JOINED:
```

```
                processMySensorJoining(ev);
                break;
            case FogEvents.SEND_PERIODIC_TUPLE:
                sendPeriodicTuple(ev);
                break;
            case FogEvents.APP_SUBMIT:
                processAppSubmit(ev);
                break;
            case FogEvents.UPDATE_NORTH_TUPLE_QUEUE:
                updateNorthTupleQueue();
                break;
            case FogEvents.UPDATE_SOUTH_TUPLE_QUEUE:
                updateSouthTupleQueue();
                break;
            case FogEvents.ACTIVE_APP_UPDATE:
                updateActiveMyApplications(ev);
                break;
            case FogEvents.ACTUATOR_JOINED:
                processMyActuatorJoined(ev);
                break;
            case FogEvents.LAUNCH_MODULE_INSTANCE:
                updateModuleInstanceCount(ev);
                break;
            case FogEvents.RESOURCE_MGMT:
                manageResources(ev);
            default:
                break;
            }
    }
```

```
	/**
	 * Perform miscellaneous resource management tasks
	 * @param ev
	 */
	private void manageResources(SimEvent ev) {
		updateEnergyConsumption();
		send(getId(), Config.RESOURCE_MGMT_INTERVAL,
FogEvents.RESOURCE_MGMT);
	}

	/**
	 * Updating the number of modules of an application module on this device
	 * @param ev instance of SimEvent containing the module and no of instances
	 */
	private void updateModuleInstanceCount(SimEvent ev) {
		ModuleLaunchConfig config = (ModuleLaunchConfig)ev.getData();
		String appId = config.getModule().getAppId();
		if(!moduleInstanceCount.containsKey(appId))
			moduleInstanceCount.put(appId, new HashMap<String,
Integer>());
		moduleInstanceCount.get(appId).put(config.getModule().getName(),
config.getInstanceCount());
		System.out.println(getName()+ " Creating
"+config.getInstanceCount()+" instances of module
"+config.getModule().getName());
	}

	private AppModule getModuleByName(String moduleName){
		AppModule module = null;
		for(Vm vm : getHost().getVmList()){
			if(((AppModule)vm).getName().equals(moduleName)){
				module=(AppModule)vm;
```

```
                        break;
                    }
                }
                return module;
            }

            /**
             * Sending periodic tuple for an application edge. Note that for multiple instances of a single source module, only one tuple is sent DOWN while instanceCount number of tuples are sent UP.
             * @param ev SimEvent instance containing the edge to send tuple on
             */
            private void sendPeriodicTuple(SimEvent ev) {
                AppEdge edge = (AppEdge)ev.getData();
                String srcModule = edge.getSource();
                AppModule module = getModuleByName(srcModule);

                if(module == null)
                    return;

                int instanceCount = module.getNumInstances();
                /*
                 * Since tuples sent through a DOWN application edge are anyways broadcasted, only UP tuples are replicated
                 */
                for(int i = 0;i<((edge.getDirection()==Tuple.UP)?instanceCount:1);i++){
                    //System.out.println(CloudSim.clock()+" : Sending periodic tuple "+edge.getTupleType());
                    Tuple tuple = applicationMap.get(module.getAppId()).createTuple(edge, getId(), module.getId());
                    updateTimingsOnSending(tuple);
```

```
                    sendToSelf(tuple);
                }
                send(getId(), edge.getPperiodicity(),
FogEvents.SEND_PERIODIC_TUPLE, edge);
        }

        protected void processMyActuatorJoined(SimEvent ev) {
                int actuatorId = ev.getSource();
                double delay = (double)ev.getData();
                getAssociatedMyActuatorIds().add(new Pair<Integer,
Double>(actuatorId, delay));
        }


        protected void updateActiveMyApplications(SimEvent ev) {
                MyApplication app = (MyApplication)ev.getData();
                getActiveMyApplications().add(app.getAppId());
        }


        public String getOperatorName(int vmId){
                for(Vm vm : this.getHost().getVmList()){
                        if(vm.getId() == vmId)
                                return ((AppModule)vm).getName();
                }
                return null;
        }

        /**
         * Update cloudet processing without scheduling future events.
         *
```

```
     * @return the double
     */
    protected double
updateCloudetProcessingWithoutSchedulingFutureEventsForce() {
        double currentTime = CloudSim.clock();
        double minTime = Double.MAX_VALUE;
        double timeDiff = currentTime - getLastProcessTime();
        double timeFrameDatacenterEnergy = 0.0;

        for (PowerHost host : this.<PowerHost> getHostList()) {
            Log.printLine();

            double time = host.updateVmsProcessing(currentTime); //
inform VMs to update processing
            if (time < minTime) {
                minTime = time;
            }

            Log.formatLine(
                    "%.2f: [Host #%d] utilization is %.2f%%",
                    currentTime,
                    host.getId(),
                    host.getUtilizationOfCpu() * 100);
        }

        if (timeDiff > 0) {
            Log.formatLine(
                    "\nEnergy consumption for the last time frame
from %.2f to %.2f:",
                    getLastProcessTime(),
                    currentTime);
```

```
                for (PowerHost host : this.<PowerHost> getHostList()) {
                        double previousUtilizationOfCpu =
host.getPreviousUtilizationOfCpu();
                        double utilizationOfCpu = host.getUtilizationOfCpu();
                        double timeFrameHostEnergy =
host.getEnergyLinearInterpolation(
                                        previousUtilizationOfCpu,
                                        utilizationOfCpu,
                                        timeDiff);
                        timeFrameDatacenterEnergy += timeFrameHostEnergy;

                        Log.printLine();
                        Log.formatLine(
                                        "%.2f: [Host #%d] utilization at %.2f
was %.2f%%, now is %.2f%%",
                                        currentTime,
                                        host.getId(),
                                        getLastProcessTime(),
                                        previousUtilizationOfCpu * 100,
                                        utilizationOfCpu * 100);
                        Log.formatLine(
                                        "%.2f: [Host #%d] energy is %.2f
W*sec",
                                        currentTime,
                                        host.getId(),
                                        timeFrameHostEnergy);
                }

                Log.formatLine(
                                "\n%.2f: Data center's energy is %.2f W*sec\n",
                                currentTime,
```

```
                                timeFrameDatacenterEnergy);
                }

                setPower(getPower() + timeFrameDatacenterEnergy);

                checkCloudletCompletion();

                /** Remove completed VMs **/
                /**
                 * Change made by HARSHIT GUPTA
                 */
                /*for (PowerHost host : this.<PowerHost> getHostList()) {
                        for (Vm vm : host.getCompletedVms()) {
                                getVmAllocationPolicy().deallocateHostForVm(vm);
                                getVmList().remove(vm);
                                Log.printLine("VM #" + vm.getId() + " has been
deallocated from host #" + host.getId());
                        }
                }*/

                Log.printLine();

                setLastProcessTime(currentTime);
                return minTime;
        }


        protected void checkCloudletCompletion() {
                boolean cloudletCompleted = false;
                List<? extends Host> list = getVmAllocationPolicy().getHostList();
                for (int i = 0; i < list.size(); i++) {
```

```
                Host host = list.get(i);
                for (Vm vm : host.getVmList()) {
                    while (vm.getCloudletScheduler().isFinishedCloudlets()) {
                        Cloudlet cl = vm.getCloudletScheduler().getNextFinishedCloudlet();
                        if (cl != null) {

                            cloudletCompleted = true;
                            Tuple tuple = (Tuple)cl;

        TimeKeeper.getInstance().tupleEndedExecution(tuple);

                            MyApplication application = getMyApplicationMap().get(tuple.getAppId());
                            Logger.debug(getName(), "Completed execution of tuple "+tuple.getCloudletId()+"on "+tuple.getDestModuleName());
                            List<Tuple> resultantTuples = application.getResultantTuples(tuple.getDestModuleName(), tuple, getId(), vm.getId());
                            for(Tuple resTuple : resultantTuples){

        resTuple.setModuleCopyMap(new HashMap<String, Integer>(tuple.getModuleCopyMap()));

        resTuple.getModuleCopyMap().put(((AppModule)vm).getName(), vm.getId());

        updateTimingsOnSending(resTuple);
                                sendToSelf(resTuple);
                            }
                            sendNow(cl.getUserId(), CloudSimTags.CLOUDLET_RETURN, cl);
                        }
                    }
                }
```

```
		}
		if(cloudletCompleted)
			updateAllocatedMips(null);
	}

	protected void updateTimingsOnSending(Tuple resTuple) {
		// TODO ADD CODE FOR UPDATING TIMINGS WHEN A TUPLE IS GENERATED FROM A PREVIOUSLY RECIEVED TUPLE.
		// WILL NEED TO CHECK IF A NEW LOOP STARTS AND INSERT A UNIQUE TUPLE ID TO IT.
		String srcModule = resTuple.getSrcModuleName();
		String destModule = resTuple.getDestModuleName();
		for(AppLoop loop : getMyApplicationMap().get(resTuple.getAppId()).getLoops()){
			if(loop.hasEdge(srcModule, destModule) && loop.isStartModule(srcModule)){
				int tupleId = TimeKeeper.getInstance().getUniqueId();
				resTuple.setActualTupleId(tupleId);

	if(!TimeKeeper.getInstance().getLoopIdToTupleIds().containsKey(loop.getLoopId()))

	TimeKeeper.getInstance().getLoopIdToTupleIds().put(loop.getLoopId(), new ArrayList<Integer>());

	TimeKeeper.getInstance().getLoopIdToTupleIds().get(loop.getLoopId()).add(tupleId);
				TimeKeeper.getInstance().getEmitTimes().put(tupleId, CloudSim.clock());

				//Logger.debug(getName(), "\tSENDING\t"+tuple.getActualTupleId()+"\tSrc:"+srcModule+"\tDest:"+destModule);

			}
```

```
                }
        }

        protected int getChildIdWithRouteTo(int targetDeviceId){
                for(Integer childId : getChildrenIds()){
                        if(targetDeviceId == childId)
                                return childId;

        if(((FogDevice)CloudSim.getEntity(childId)).getChildIdWithRouteTo(targetD
eviceId) != -1)
                                return childId;
                }
                return -1;
        }

        protected int getChildIdForTuple(Tuple tuple){
                if(tuple.getDirection() == Tuple.ACTUATOR){
                        int gatewayId =
((MyActuator)CloudSim.getEntity(tuple.getActuatorId())).getGatewayDeviceId();
                        return getChildIdWithRouteTo(gatewayId);
                }
                return -1;
        }

        protected void updateAllocatedMips(String incomingOperator){
                getHost().getVmScheduler().deallocatePesForAllVms();
                for(final Vm vm : getHost().getVmList()){
                        if(vm.getCloudletScheduler().runningCloudlets() > 0 ||
((AppModule)vm).getName().equals(incomingOperator)){
                                getHost().getVmScheduler().allocatePesForVm(vm,
new ArrayList<Double>(){
```

```
                                        protected static final long serialVersionUID =
1L;
                                {add((double) getHost().getTotalMips());}});
                        }else{
                                getHost().getVmScheduler().allocatePesForVm(vm,
new ArrayList<Double>(){
                                        protected static final long serialVersionUID =
1L;
                                {add(0.0);}});
                        }
                }

                updateEnergyConsumption();

        }

        private void updateEnergyConsumption() {
                double totalMipsAllocated = 0;
                for(final Vm vm : getHost().getVmList()){
                        AppModule operator = (AppModule)vm;
                        operator.updateVmProcessing(CloudSim.clock(),
getVmAllocationPolicy().getHost(operator).getVmScheduler()
                                        .getAllocatedMipsForVm(operator));
                        totalMipsAllocated +=
getHost().getTotalAllocatedMipsForVm(vm);
                }

                double timeNow = CloudSim.clock();
                double currentEnergyConsumption = getEnergyConsumption();
                double newEnergyConsumption = currentEnergyConsumption +
(timeNow-
lastUtilizationUpdateTime)*getHost().getPowerModel().getPower(lastUtilization);
                setEnergyConsumption(newEnergyConsumption);
```

```
		/*if(getName().equals("d-0")){
			System.out.println("------------------------");
			System.out.println("Utilization = "+lastUtilization);
			System.out.println("Power =
"+getHost().getPowerModel().getPower(lastUtilization));
			System.out.println(timeNow-lastUtilizationUpdateTime);
		}*/

		double currentCost = getTotalCost();
		double newcost = currentCost + (timeNow-
lastUtilizationUpdateTime)*getRatePerMips()*lastUtilization*getHost().getTotalMip
s();
		setTotalCost(newcost);

		lastUtilization = Math.min(1,
totalMipsAllocated/getHost().getTotalMips());
		lastUtilizationUpdateTime = timeNow;
	}

	protected void processAppSubmit(SimEvent ev) {
		MyApplication app = (MyApplication)ev.getData();
		applicationMap.put(app.getAppId(), app);
	}

	protected void addChild(int childId){

	if(CloudSim.getEntityName(childId).toLowerCase().contains("sensor"))
			return;
		if(!getChildrenIds().contains(childId) && childId != getId())
			getChildrenIds().add(childId);
		if(!getChildToOperatorsMap().containsKey(childId))
```

```
                getChildToOperatorsMap().put(childId, new
ArrayList<String>());
        }

        protected void updateCloudTraffic(){
                int time = (int)CloudSim.clock()/1000;
                if(!cloudTrafficMap.containsKey(time))
                        cloudTrafficMap.put(time, 0);
                cloudTrafficMap.put(time, cloudTrafficMap.get(time)+1);
        }

        protected void sendTupleToMyActuator(Tuple tuple){
                /*for(Pair<Integer, Double> actuatorAssociation :
getAssociatedMyActuatorIds()){
                        int actuatorId = actuatorAssociation.getFirst();
                        double delay = actuatorAssociation.getSecond();
                        if(actuatorId == tuple.getMyActuatorId()){
                                send(actuatorId, delay, FogEvents.TUPLE_ARRIVAL,
tuple);
                                return;
                        }
                }
                int childId = getChildIdForTuple(tuple);
                if(childId != -1)
                        sendDown(tuple, childId);*/
                for(Pair<Integer, Double> actuatorAssociation :
getAssociatedMyActuatorIds()){
                        int actuatorId = actuatorAssociation.getFirst();
                        double delay = actuatorAssociation.getSecond();
                        String actuatorType =
((MyActuator)CloudSim.getEntity(actuatorId)).getMyActuatorType();
                        if(tuple.getDestModuleName().equals(actuatorType)){
```

```
                                send(actuatorId, delay, FogEvents.TUPLE_ARRIVAL,
tuple);
                                return;
                        }
                }
                for(int childId : getChildrenIds()){
                        sendDown(tuple, childId);
                }
        }
        int numClients=0;
        protected void processTupleArrival(SimEvent ev){
                Tuple tuple = (Tuple)ev.getData();

                if(getName().equals("cloud")){
                        updateCloudTraffic();
                }

                /*if(getName().equals("d-0") &&
tuple.getTupleType().equals("_SENSOR")){
                        System.out.println(++numClients);
                }*/
                Logger.debug(getName(), "Received tuple
"+tuple.getCloudletId()+"with tupleType = "+tuple.getTupleType()+"\t| Source : "+
                CloudSim.getEntityName(ev.getSource())+"|Dest :
"+CloudSim.getEntityName(ev.getDestination()));
                send(ev.getSource(), CloudSim.getMinTimeBetweenEvents(),
FogEvents.TUPLE_ACK);

        if(FogUtils.appIdToGeoCoverageMap.containsKey(tuple.getAppId())){
                }
```

```
		if(tuple.getDirection() == Tuple.ACTUATOR){
			sendTupleToMyActuator(tuple);
			return;
		}

		if(getHost().getVmList().size() > 0){
			final AppModule operator = (AppModule)getHost().getVmList().get(0);
			if(CloudSim.clock() > 0){

	getHost().getVmScheduler().deallocatePesForVm(operator);

	getHost().getVmScheduler().allocatePesForVm(operator, new ArrayList<Double>(){
					protected static final long serialVersionUID = 1L;
				{add((double) getHost().getTotalMips());}});
			}
		}

		if(getName().equals("cloud") && tuple.getDestModuleName()==null){
			sendNow(getControllerId(), FogEvents.TUPLE_FINISHED, null);
		}

		if(appToModulesMap.containsKey(tuple.getAppId())){

	if(appToModulesMap.get(tuple.getAppId()).contains(tuple.getDestModuleName())){
				int vmId = -1;
				for(Vm vm : getHost().getVmList()){

	if(((AppModule)vm).getName().equals(tuple.getDestModuleName())){
```

```
                                        vmId = vm.getId();
                                }
                        }
                        if(vmId < 0
                                                ||
(tuple.getModuleCopyMap().containsKey(tuple.getDestModuleName()) &&

        tuple.getModuleCopyMap().get(tuple.getDestModuleName())!=vmId )){
                                return;
                        }
                        tuple.setVmId(vmId);
                        //Logger.error(getName(), "Executing tuple for operator
" + moduleName);

                        updateTimingsOnReceipt(tuple);

                        executeTuple(ev, tuple.getDestModuleName());
                }else if(tuple.getDestModuleName()!=null){
                        if(tuple.getDirection() == Tuple.UP)
                                sendUp(tuple);
                        else if(tuple.getDirection() == Tuple.DOWN){
                                for(int childId : getChildrenIds())
                                        sendDown(tuple, childId);
                        }
                }else{
                        sendUp(tuple);
                }
        }else{
                if(tuple.getDirection() == Tuple.UP)
                        sendUp(tuple);
                else if(tuple.getDirection() == Tuple.DOWN){
```

```
                        for(int childId : getChildrenIds())
                            sendDown(tuple, childId);
                    }
                }
            }

            protected void updateTimingsOnReceipt(Tuple tuple) {
                MyApplication app = getMyApplicationMap().get(tuple.getAppId());
                String srcModule = tuple.getSrcModuleName();
                String destModule = tuple.getDestModuleName();
                List<AppLoop> loops = app.getLoops();
                for(AppLoop loop : loops){
                    if(loop.hasEdge(srcModule, destModule) &&
loop.isEndModule(destModule)){
                        Double startTime =
TimeKeeper.getInstance().getEmitTimes().get(tuple.getActualTupleId());
                        if(startTime==null)
                            break;

            if(!TimeKeeper.getInstance().getLoopIdToCurrentAverage().containsKey(loo
p.getLoopId())){

            TimeKeeper.getInstance().getLoopIdToCurrentAverage().put(loop.getLoopId(
), 0.0);

            TimeKeeper.getInstance().getLoopIdToCurrentNum().put(loop.getLoopId(),
0);
                        }
                        double currentAverage =
TimeKeeper.getInstance().getLoopIdToCurrentAverage().get(loop.getLoopId());
                        int currentCount =
TimeKeeper.getInstance().getLoopIdToCurrentNum().get(loop.getLoopId());
                        double delay = CloudSim.clock()-
TimeKeeper.getInstance().getEmitTimes().get(tuple.getActualTupleId());
```

```
TimeKeeper.getInstance().getEmitTimes().remove(tuple.getActualTupleId());
                        double newAverage = (currentAverage*currentCount + delay)/(currentCount+1);

TimeKeeper.getInstance().getLoopIdToCurrentAverage().put(loop.getLoopId(), newAverage);

TimeKeeper.getInstance().getLoopIdToCurrentNum().put(loop.getLoopId(), currentCount+1);
                        break;
                    }
                }
            }

protected void processMySensorJoining(SimEvent ev){
            send(ev.getSource(), CloudSim.getMinTimeBetweenEvents(), FogEvents.TUPLE_ACK);
        }

protected void executeTuple(SimEvent ev, String moduleName){
            Logger.debug(getName(), "Executing tuple on module "+moduleName);
            Tuple tuple = (Tuple)ev.getData();

            AppModule module = getModuleByName(moduleName);

            if(tuple.getDirection() == Tuple.UP){
                String srcModule = tuple.getSrcModuleName();

if(!module.getDownInstanceIdsMaps().containsKey(srcModule))
                    module.getDownInstanceIdsMaps().put(srcModule, new ArrayList<Integer>());
```

```
if(!module.getDownInstanceIdsMaps().get(srcModule).contains(tuple.getSourceModuleId()))

module.getDownInstanceIdsMaps().get(srcModule).add(tuple.getSourceModuleId());

                int instances = -1;
                for(String _moduleName : module.getDownInstanceIdsMaps().keySet()){
                        instances = Math.max(module.getDownInstanceIdsMaps().get(_moduleName).size(), instances);
                }
                module.setNumInstances(instances);
        }

        TimeKeeper.getInstance().tupleStartedExecution(tuple);
        updateAllocatedMips(moduleName);
        processCloudletSubmit(ev, false);
        updateAllocatedMips(moduleName);
        /*for(Vm vm : getHost().getVmList()){
                Logger.error(getName(), "MIPS allocated to "+((AppModule)vm).getName()+" = "+getHost().getTotalAllocatedMipsForVm(vm));
        }*/
    }

    protected void processModuleArrival(SimEvent ev){
        AppModule module = (AppModule)ev.getData();
        String appId = module.getAppId();
        if(!appToModulesMap.containsKey(appId)){
                appToModulesMap.put(appId, new ArrayList<String>());
        }
```

```
                appToModulesMap.get(appId).add(module.getName());
                processVmCreate(ev, false);
                if (module.isBeingInstantiated()) {
                        module.setBeingInstantiated(false);
                }

                initializePeriodicTuples(module);

                module.updateVmProcessing(CloudSim.clock(),
getVmAllocationPolicy().getHost(module).getVmScheduler()
                                .getAllocatedMipsForVm(module));
                System.out.println(CloudSim.clock()+" "+module.getName() + "
Launched in "+getName());
        }

        private void initializePeriodicTuples(AppModule module) {
                String appId = module.getAppId();
                MyApplication app = getMyApplicationMap().get(appId);
                List<AppEdge> periodicEdges =
app.getPeriodicEdges(module.getName());
                for(AppEdge edge : periodicEdges){
                        send(getId(), edge.getPeriodicity(),
FogEvents.SEND_PERIODIC_TUPLE, edge);
                }
        }

        protected void processOperatorRelease(SimEvent ev){
                this.processVmMigrate(ev, false);
        }


        protected void updateNorthTupleQueue(){
```

```
            if(!getNorthTupleQueue().isEmpty()){
                Tuple tuple = getNorthTupleQueue().poll();
                sendUpFreeLink(tuple);
            }else{
                setNorthLinkBusy(false);
            }
        }

        protected void sendUpFreeLink(Tuple tuple){
            double networkDelay =
tuple.getCloudletFileSize()/getUplinkBandwidth();
            setNorthLinkBusy(true);
            send(getId(), networkDelay,
FogEvents.UPDATE_NORTH_TUPLE_QUEUE);
            send(parentId, networkDelay+getUplinkLatency(),
FogEvents.TUPLE_ARRIVAL, tuple);
            NetworkUsageMonitor.sendingTuple(getUplinkLatency(),
tuple.getCloudletFileSize());
        }

        protected void sendUp(Tuple tuple){
            if(parentId > 0){
                if(!isNorthLinkBusy()){
                    sendUpFreeLink(tuple);
                }else{
                    northTupleQueue.add(tuple);
                }
            }
        }

        protected void updateSouthTupleQueue(){
```

```
		if(!getSouthTupleQueue().isEmpty()){
			Pair<Tuple, Integer> pair = getSouthTupleQueue().poll();
			sendDownFreeLink(pair.getFirst(), pair.getSecond());
		}else{
			setSouthLinkBusy(false);
		}
	}

	protected void sendDownFreeLink(Tuple tuple, int childId){
		double networkDelay = tuple.getCloudletFileSize()/getDownlinkBandwidth();
		//Logger.debug(getName(), "Sending tuple with tupleType = "+tuple.getTupleType()+" DOWN");
		setSouthLinkBusy(true);
		double latency = getChildToLatencyMap().get(childId);
		send(getId(), networkDelay, FogEvents.UPDATE_SOUTH_TUPLE_QUEUE);
		send(childId, networkDelay+latency, FogEvents.TUPLE_ARRIVAL, tuple);
		NetworkUsageMonitor.sendingTuple(latency, tuple.getCloudletFileSize());
	}

	protected void sendDown(Tuple tuple, int childId){
		if(getChildrenIds().contains(childId)){
			if(!isSouthLinkBusy()){
				sendDownFreeLink(tuple, childId);
			}else{
				southTupleQueue.add(new Pair<Tuple, Integer>(tuple, childId));
			}
		}
```

```
	}



	protected void sendToSelf(Tuple tuple){
		send(getId(), CloudSim.getMinTimeBetweenEvents(),
FogEvents.TUPLE_ARRIVAL, tuple);
	}
	public PowerHost getHost(){
		return (PowerHost) getHostList().get(0);
	}
	public int getParentId() {
		return parentId;
	}
	public void setParentId(int parentId) {
		this.parentId = parentId;
	}
	public List<Integer> getChildrenIds() {
		return childrenIds;
	}
	public void setChildrenIds(List<Integer> childrenIds) {
		this.childrenIds = childrenIds;
	}
	public double getUplinkBandwidth() {
		return uplinkBandwidth;
	}
	public void setUplinkBandwidth(double uplinkBandwidth) {
		this.uplinkBandwidth = uplinkBandwidth;
	}
	public double getUplinkLatency() {
		return uplinkLatency;
	}
```

```
public void setUplinkLatency(double uplinkLatency) {
	this.uplinkLatency = uplinkLatency;
}
public boolean isSouthLinkBusy() {
	return isSouthLinkBusy;
}
public boolean isNorthLinkBusy() {
	return isNorthLinkBusy;
}
public void setSouthLinkBusy(boolean isSouthLinkBusy) {
	this.isSouthLinkBusy = isSouthLinkBusy;
}
public void setNorthLinkBusy(boolean isNorthLinkBusy) {
	this.isNorthLinkBusy = isNorthLinkBusy;
}
public int getControllerId() {
	return controllerId;
}
public void setControllerId(int controllerId) {
	this.controllerId = controllerId;
}
public List<String> getActiveMyApplications() {
	return activeMyApplications;
}
public void setActiveMyApplications(List<String> activeMyApplications) {
	this.activeMyApplications = activeMyApplications;
}
public Map<Integer, List<String>> getChildToOperatorsMap() {
	return childToOperatorsMap;
}
```

```
	public void setChildToOperatorsMap(Map<Integer, List<String>>
childToOperatorsMap) {
		this.childToOperatorsMap = childToOperatorsMap;
	}

	public Map<String, MyApplication> getMyApplicationMap() {
		return applicationMap;
	}

	public void setMyApplicationMap(Map<String, MyApplication>
applicationMap) {
		this.applicationMap = applicationMap;
	}

	public Queue<Tuple> getNorthTupleQueue() {
		return northTupleQueue;
	}

	public void setNorthTupleQueue(Queue<Tuple> northTupleQueue) {
		this.northTupleQueue = northTupleQueue;
	}

	public Queue<Pair<Tuple, Integer>> getSouthTupleQueue() {
		return southTupleQueue;
	}

	public void setSouthTupleQueue(Queue<Pair<Tuple, Integer>>
southTupleQueue) {
		this.southTupleQueue = southTupleQueue;
	}
```

```
	public double getDownlinkBandwidth() {
		return downlinkBandwidth;
	}

	public void setDownlinkBandwidth(double downlinkBandwidth) {
		this.downlinkBandwidth = downlinkBandwidth;
	}

	public List<Pair<Integer, Double>> getAssociatedMyActuatorIds() {
		return associatedMyActuatorIds;
	}

	public void setAssociatedMyActuatorIds(List<Pair<Integer, Double>>
associatedMyActuatorIds) {
		this.associatedMyActuatorIds = associatedMyActuatorIds;
	}

	public double getEnergyConsumption() {
		return energyConsumption;
	}

	public void setEnergyConsumption(double energyConsumption) {
		this.energyConsumption = energyConsumption;
	}
	public Map<Integer, Double> getChildToLatencyMap() {
		return childToLatencyMap;
	}

	public void setChildToLatencyMap(Map<Integer, Double>
childToLatencyMap) {
		this.childToLatencyMap = childToLatencyMap;
```

```
}

public int getLevel() {
        return level;
}

public void setLevel(int level) {
        this.level = level;
}

public double getRatePerMips() {
        return ratePerMips;
}

public void setRatePerMips(double ratePerMips) {
        this.ratePerMips = ratePerMips;
}
public double getTotalCost() {
        return totalCost;
}

public void setTotalCost(double totalCost) {
        this.totalCost = totalCost;
}

public Map<String, Map<String, Integer>> getModuleInstanceCount() {
        return moduleInstanceCount;
}

public void setModuleInstanceCount(
```

```
                    Map<String, Map<String, Integer>> moduleInstanceCount) {
            this.moduleInstanceCount = moduleInstanceCount;
        }

        ////////////////////////////////////////
        private int mips;

        public int getMips() {
            return mips;
        }

        public void setMips(int mips) {
            this.mips = mips;
        }

}
```

MyController.java

```
package org.fog.placement;

import java.util.Calendar;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.AppModule;
import org.fog.application.MyApplication;
import org.fog.entities.MyActuator;
import org.fog.entities.MyFogDevice;
import org.fog.entities.MySensor;
import org.fog.utils.Config;
import org.fog.utils.FogEvents;
import org.fog.utils.FogUtils;
import org.fog.utils.NetworkUsageMonitor;
import org.fog.utils.TimeKeeper;

public class MyController extends SimEntity{

	public static boolean ONLY_CLOUD = false;

	private List<MyFogDevice> fogDevices;
	private List<MySensor> sensors;
	private List<MyActuator> actuators;

	private Map<String, MyApplication> applications;
```

```
	private Map<String, Integer> appLaunchDelays;

	private Map<String, MyModulePlacement> appModulePlacementPolicy;

	public MyController(String name, List<MyFogDevice> fogDevices, List<MySensor> sensors, List<MyActuator> actuators) {
		super(name);
		this.applications = new HashMap<String, MyApplication>();
		setAppLaunchDelays(new HashMap<String, Integer>());
		setAppModulePlacementPolicy(new HashMap<String, MyModulePlacement>());
		for(MyFogDevice fogDevice : fogDevices){
			fogDevice.setControllerId(getId());
		}
		setMyFogDevices(fogDevices);
		setMyActuators(actuators);
		setMySensors(sensors);
		connectWithLatencies();
	}

	private MyFogDevice getMyFogDeviceById(int id){
		for(MyFogDevice fogDevice : getMyFogDevices()){
			if(id==fogDevice.getId())
				return fogDevice;
		}
		return null;
	}

	private void connectWithLatencies(){
		for(MyFogDevice fogDevice : getMyFogDevices()){
			MyFogDevice parent = getMyFogDeviceById(fogDevice.getParentId());
			if(parent == null)
				continue;
```

```
                double latency = fogDevice.getUplinkLatency();
                parent.getChildToLatencyMap().put(fogDevice.getId(),
latency);
                parent.getChildrenIds().add(fogDevice.getId());
            }
        }

        @Override
        public void startEntity() {
            for(String appId : applications.keySet()){
                if(getAppLaunchDelays().get(appId)==0)
                    processAppSubmit(applications.get(appId));
                else
                    send(getId(),          getAppLaunchDelays().get(appId),
FogEvents.APP_SUBMIT, applications.get(appId));
            }

            send(getId(),          Config.RESOURCE_MANAGE_INTERVAL,
FogEvents.CONTROLLER_RESOURCE_MANAGE);

            send(getId(),                  Config.MAX_SIMULATION_TIME,
FogEvents.STOP_SIMULATION);

            for(MyFogDevice dev : getMyFogDevices())
                sendNow(dev.getId(), FogEvents.RESOURCE_MGMT);

        }

        @Override
        public void processEvent(SimEvent ev) {
            switch(ev.getTag()){
            case FogEvents.APP_SUBMIT:
                processAppSubmit(ev);
                break;
```

```
                case FogEvents.TUPLE_FINISHED:
                        processTupleFinished(ev);
                        break;
                case FogEvents.CONTROLLER_RESOURCE_MANAGE:
                        manageResources();
                        break;
                case FogEvents.STOP_SIMULATION:
                        CloudSim.stopSimulation();
                        printTimeDetails();
                        printPowerDetails();
                        printCostDetails();
                        printNetworkUsageDetails();
                        System.exit(0);
                        break;

                }
        }

        private void printNetworkUsageDetails() {
                System.out.println("Total network usage =
"+NetworkUsageMonitor.getNetworkUsage()/Config.MAX_SIMULATION_TIME);

        }

        private MyFogDevice getCloud(){
                for(MyFogDevice dev : getMyFogDevices())
                        if(dev.getName().equals("cloud"))
                                return dev;
                return null;
        }

        private void printCostDetails(){
                System.out.println("Cost of execution in cloud =
"+getCloud().getTotalCost());
```

```
	}

	private void printPowerDetails() {
		for(MyFogDevice fogDevice : getMyFogDevices()){
			System.out.println(fogDevice.getName() + " : Energy
Consumed = "+fogDevice.getEnergyConsumption());
		}
	}

	private String getStringForLoopId(int loopId){
		for(String appId : getMyApplications().keySet()){
			MyApplication app = getMyApplications().get(appId);
			for(AppLoop loop : app.getLoops()){
				if(loop.getLoopId() == loopId)
					return loop.getModules().toString();
			}
		}
		return null;
	}
	private void printTimeDetails() {

	System.out.println("=========================================
");
		System.out.println("==============                    RESULTS
==================");

	System.out.println("=========================================
");
		System.out.println("EXECUTION TIME : "+
(Calendar.getInstance().getTimeInMillis() -
TimeKeeper.getInstance().getSimulationStartTime()));

	System.out.println("=========================================
");
```

```
                System.out.println("APPLICATION LOOP DELAYS");

        System.out.println("========================================
");
                for(Integer                loopId                :
TimeKeeper.getInstance().getLoopIdToTupleIds().keySet()){

                        System.out.println(getStringForLoopId(loopId)    +    "    --->
"+TimeKeeper.getInstance().getLoopIdToCurrentAverage().get(loopId));
                }

        System.out.println("========================================
");
                System.out.println("TUPLE CPU EXECUTION DELAY");

        System.out.println("========================================
");

                for(String                tupleType                :
TimeKeeper.getInstance().getTupleTypeToAverageCpuTime().keySet()){
                        System.out.println(tupleType    +    "    --->
"+TimeKeeper.getInstance().getTupleTypeToAverageCpuTime().get(tupleType));
                }


        System.out.println("========================================
");
        }

        protected void manageResources(){
                send(getId(),          Config.RESOURCE_MANAGE_INTERVAL,
FogEvents.CONTROLLER_RESOURCE_MANAGE);
        }
```

```
	private void processTupleFinished(SimEvent ev) {
	}

	@Override
	public void shutdownEntity() {
	}

	public void submitApplication(MyApplication application, int delay, MyModulePlacement modulePlacement){
		FogUtils.appIdToGeoCoverageMap.put(application.getAppId(), application.getGeoCoverage());
		getMyApplications().put(application.getAppId(), application);
		getAppLaunchDelays().put(application.getAppId(), delay);
		getAppModulePlacementPolicy().put(application.getAppId(), modulePlacement);

		for(MySensor sensor : sensors){
			sensor.setApp(getMyApplications().get(sensor.getAppId()));
		}
		for(MyActuator ac : actuators){
			ac.setApp(getMyApplications().get(ac.getAppId()));
		}

		for(AppEdge edge : application.getEdges()){
			if(edge.getEdgeType() == AppEdge.ACTUATOR){
				String moduleName = edge.getSource();
				for(MyActuator actuator : getMyActuators()){

	if(actuator.getMyActuatorType().equalsIgnoreCase(edge.getDestination()))

	application.getModuleByName(moduleName).subscribeActuator(actuator.getId(), edge.getTupleType());
				}
			}
```

```
            }
        }

        public void submitApplication(MyApplication application,
MyModulePlacement modulePlacement){
            submitApplication(application, 0, modulePlacement);
        }


        private void processAppSubmit(SimEvent ev){
            MyApplication app = (MyApplication) ev.getData();
            processAppSubmit(app);
        }

        private void processAppSubmit(MyApplication application){
            System.out.println(CloudSim.clock()+" Submitted application "+
application.getAppId());
            FogUtils.appIdToGeoCoverageMap.put(application.getAppId(),
application.getGeoCoverage());
            getMyApplications().put(application.getAppId(), application);

            MyModulePlacement modulePlacement =
getAppModulePlacementPolicy().get(application.getAppId());
            for(MyFogDevice fogDevice : fogDevices){
                sendNow(fogDevice.getId(),
FogEvents.ACTIVE_APP_UPDATE, application);
            }

            Map<Integer, List<AppModule>> deviceToModuleMap =
modulePlacement.getDeviceToModuleMap();


            for(Integer deviceId : deviceToModuleMap.keySet()){
                for(AppModule module : deviceToModuleMap.get(deviceId)){
```

```
                                sendNow(deviceId, FogEvents.APP_SUBMIT, application);
                                System.out.println(CloudSim.clock()+" Trying to Launch "+ module.getName() + " in "+getMyFogDeviceById(deviceId).getName());
                                sendNow(deviceId, FogEvents.LAUNCH_MODULE, module);
                        }
                }

        }

        public List<MyFogDevice> getMyFogDevices() {
                return fogDevices;
        }

        public void setMyFogDevices(List<MyFogDevice> fogDevices) {
                this.fogDevices = fogDevices;
        }

        public Map<String, Integer> getAppLaunchDelays() {
                return appLaunchDelays;
        }

        public void setAppLaunchDelays(Map<String, Integer> appLaunchDelays) {
                this.appLaunchDelays = appLaunchDelays;
        }

        public Map<String, MyApplication> getMyApplications() {
                return applications;
        }

        public void setMyApplications(Map<String, MyApplication> applications) {
                this.applications = applications;
```

```
	}

	public List<MySensor> getMySensors() {
		return sensors;
	}

	public void setMySensors(List<MySensor> sensors) {
		for(MySensor sensor : sensors)
			sensor.setControllerId(getId());
		this.sensors = sensors;
	}

	public List<MyActuator> getMyActuators() {
		return actuators;
	}

	public void setMyActuators(List<MyActuator> actuators) {
		this.actuators = actuators;
	}

	public Map<String, MyModulePlacement> getAppModulePlacementPolicy() {
		return appModulePlacementPolicy;
	}

	public void setAppModulePlacementPolicy(Map<String,
MyModulePlacement> appModulePlacementPolicy) {
		this.appModulePlacementPolicy = appModulePlacementPolicy;
	}
}
```

MySensor.java

```
package org.fog.entities;

import java.util.ArrayList;

import org.cloudbus.cloudsim.UtilizationModelFull;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.fog.application.AppEdge;
import org.fog.application.AppLoop;
import org.fog.application.MyApplication;
import org.fog.utils.FogEvents;
import org.fog.utils.FogUtils;
import org.fog.utils.GeoLocation;
import org.fog.utils.Logger;
import org.fog.utils.TimeKeeper;
import org.fog.utils.distribution.Distribution;

public class MySensor extends SimEntity{

	private int gatewayDeviceId;
	private GeoLocation geoLocation;
	private long outputSize;
	private String appId;
	private int userId;
	private String tupleType;
	private String sensorName;
	private String destModuleName;
	private Distribution transmitDistribution;
	private int controllerId;
	private MyApplication app;
	private double latency;
```

```
	public MySensor(String name, int userId, String appId, int gatewayDeviceId,
double latency, GeoLocation geoLocation,
			Distribution transmitDistribution, int cpuLength, int nwLength,
String tupleType, String destModuleName) {
		super(name);
		this.setAppId(appId);
		this.gatewayDeviceId = gatewayDeviceId;
		this.geoLocation = geoLocation;
		this.outputSize = 3;
		this.setTransmitDistribution(transmitDistribution);
		setUserId(userId);
		setDestModuleName(destModuleName);
		setTupleType(tupleType);
		setSensorName(sensorName);
		setLatency(latency);
	}

	public MySensor(String name, int userId, String appId, int gatewayDeviceId,
double latency, GeoLocation geoLocation,
			Distribution transmitDistribution, String tupleType) {
		super(name);
		this.setAppId(appId);
		this.gatewayDeviceId = gatewayDeviceId;
		this.geoLocation = geoLocation;
		this.outputSize = 3;
		this.setTransmitDistribution(transmitDistribution);
		setUserId(userId);
		setTupleType(tupleType);
		setSensorName(sensorName);
		setLatency(latency);
	}

	/**
```

```
	 * This constructor is called from the code that generates PhysicalTopology from JSON
	 * @param name
	 * @param tupleType
	 * @param string
	 * @param userId
	 * @param appId
	 * @param transmitDistribution
	 */
	public MySensor(String name, String tupleType, int userId, String appId, Distribution transmitDistribution) {
		super(name);
		this.setAppId(appId);
		this.setTransmitDistribution(transmitDistribution);
		setTupleType(tupleType);
		setSensorName(tupleType);
		setUserId(userId);
	}

	public void transmit(){
		AppEdge _edge = null;
		for(AppEdge edge : getApp().getEdges()){
			if(edge.getSource().equals(getTupleType()))
				_edge = edge;
		}
		long cpuLength = (long) _edge.getTupleCpuLength();
		long nwLength = (long) _edge.getTupleNwLength();

		Tuple tuple = new Tuple(getAppId(), FogUtils.generateTupleId(), Tuple.UP, cpuLength, 1, nwLength, outputSize,
				new UtilizationModelFull(), new UtilizationModelFull(), new UtilizationModelFull());
		tuple.setUserId(getUserId());
		tuple.setTupleType(getTupleType());
```

```
                tuple.setDestModuleName(_edge.getDestination());
                tuple.setSrcModuleName(getSensorName());
                Logger.debug(getName(), "Sending tuple with tupleId = "+tuple.getCloudletId());

                int actualTupleId = updateTimings(getSensorName(), tuple.getDestModuleName());
                tuple.setActualTupleId(actualTupleId);

                send(gatewayDeviceId, getLatency(), FogEvents.TUPLE_ARRIVAL,tuple);
        }

        private int updateTimings(String src, String dest){
                MyApplication application = getApp();
                for(AppLoop loop : application.getLoops()){
                        if(loop.hasEdge(src, dest)){

                                int tupleId = TimeKeeper.getInstance().getUniqueId();

        if(!TimeKeeper.getInstance().getLoopIdToTupleIds().containsKey(loop.getLoopId()))

        TimeKeeper.getInstance().getLoopIdToTupleIds().put(loop.getLoopId(), new ArrayList<Integer>());

        TimeKeeper.getInstance().getLoopIdToTupleIds().get(loop.getLoopId()).add(tupleId);
                                TimeKeeper.getInstance().getEmitTimes().put(tupleId, CloudSim.clock());
                                return tupleId;
                        }
                }
```

```
            return -1;
        }

        @Override
        public void startEntity() {
            send(gatewayDeviceId, CloudSim.getMinTimeBetweenEvents(),
FogEvents.SENSOR_JOINED, geoLocation);
            send(getId(), getTransmitDistribution().getNextValue(),
FogEvents.EMIT_TUPLE);
        }

        @Override
        public void processEvent(SimEvent ev) {
            switch(ev.getTag()){
            case FogEvents.TUPLE_ACK:
                //transmit(transmitDistribution.getNextValue());
                break;
            case FogEvents.EMIT_TUPLE:
                transmit();
                send(getId(), getTransmitDistribution().getNextValue(),
FogEvents.EMIT_TUPLE);
                break;
            }

        }

        @Override
        public void shutdownEntity() {

        }

        public int getGatewayDeviceId() {
            return gatewayDeviceId;
        }
```

```
public void setGatewayDeviceId(int gatewayDeviceId) {
	this.gatewayDeviceId = gatewayDeviceId;
}

public GeoLocation getGeoLocation() {
	return geoLocation;
}

public void setGeoLocation(GeoLocation geoLocation) {
	this.geoLocation = geoLocation;
}

public int getUserId() {
	return userId;
}

public void setUserId(int userId) {
	this.userId = userId;
}

public String getTupleType() {
	return tupleType;
}

public void setTupleType(String tupleType) {
	this.tupleType = tupleType;
}

public String getSensorName() {
	return sensorName;
}

public void setSensorName(String sensorName) {
```

```
        this.sensorName = sensorName;
}

public String getAppId() {
        return appId;
}

public void setAppId(String appId) {
        this.appId = appId;
}

public String getDestModuleName() {
        return destModuleName;
}

public void setDestModuleName(String destModuleName) {
        this.destModuleName = destModuleName;
}

public Distribution getTransmitDistribution() {
        return transmitDistribution;
}

public void setTransmitDistribution(Distribution transmitDistribution) {
        this.transmitDistribution = transmitDistribution;
}

public int getControllerId() {
        return controllerId;
}

public void setControllerId(int controllerId) {
        this.controllerId = controllerId;
}
```

```
	public MyApplication getApp() {
		return app;
	}

	public void setApp(MyApplication app) {
		this.app = app;
	}

	public Double getLatency() {
		return latency;
	}

	public void setLatency(Double latency) {
		this.latency = latency;
	}

}
```

MyActuator.java

```
package org.fog.entities;

import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.core.SimEntity;
import org.cloudbus.cloudsim.core.SimEvent;
import org.fog.application.AppLoop;
import org.fog.application.MyApplication;
import org.fog.utils.FogEvents;
import org.fog.utils.GeoLocation;
import org.fog.utils.Logger;
import org.fog.utils.TimeKeeper;

public class MyActuator extends SimEntity{

	private int gatewayDeviceId;
	private double latency;
	private GeoLocation geoLocation;
	private String appId;
	private int userId;
	private String actuatorType;
	private MyApplication app;

	public MyActuator(String name, int userId, String appId, int gatewayDeviceId, double latency, GeoLocation geoLocation, String actuatorType, String srcModuleName) {
		super(name);
		this.setAppId(appId);
		this.gatewayDeviceId = gatewayDeviceId;
		this.geoLocation = geoLocation;
		setUserId(userId);
		setMyActuatorType(actuatorType);
		setLatency(latency);
```

```
	}

	public MyActuator(String name, int userId, String appId, String actuatorType)
{
		super(name);
		this.setAppId(appId);
		setUserId(userId);
		setMyActuatorType(actuatorType);
	}

	@Override
	public void startEntity() {
		sendNow(gatewayDeviceId, FogEvents.ACTUATOR_JOINED,
getLatency());
	}

	@Override
	public void processEvent(SimEvent ev) {
		switch(ev.getTag()){
		case FogEvents.TUPLE_ARRIVAL:
			processTupleArrival(ev);
			break;
		}
	}

	private void processTupleArrival(SimEvent ev) {
		Tuple tuple = (Tuple)ev.getData();
		Logger.debug(getName(), "Received tuple "+tuple.getCloudletId()+"on
"+tuple.getDestModuleName());
		String srcModule = tuple.getSrcModuleName();
		String destModule = tuple.getDestModuleName();
		MyApplication app = getApp();

		for(AppLoop loop : app.getLoops()){
```

```
                    if(loop.hasEdge(srcModule, destModule) &&
loop.isEndModule(destModule)){

                        Double startTime =
TimeKeeper.getInstance().getEmitTimes().get(tuple.getActualTupleId());
                        if(startTime==null)
                            break;

        if(!TimeKeeper.getInstance().getLoopIdToCurrentAverage().containsKey(loo
p.getLoopId())){

        TimeKeeper.getInstance().getLoopIdToCurrentAverage().put(loop.getLoopId(
), 0.0);

        TimeKeeper.getInstance().getLoopIdToCurrentNum().put(loop.getLoopId(),
0);
                        }
                        double currentAverage =
TimeKeeper.getInstance().getLoopIdToCurrentAverage().get(loop.getLoopId());
                        int currentCount =
TimeKeeper.getInstance().getLoopIdToCurrentNum().get(loop.getLoopId());
                        double delay = CloudSim.clock()-
TimeKeeper.getInstance().getEmitTimes().get(tuple.getActualTupleId());

        TimeKeeper.getInstance().getEmitTimes().remove(tuple.getActualTupleId());
                        double newAverage = (currentAverage*currentCount +
delay)/(currentCount+1);

        TimeKeeper.getInstance().getLoopIdToCurrentAverage().put(loop.getLoopId(
), newAverage);

        TimeKeeper.getInstance().getLoopIdToCurrentNum().put(loop.getLoopId(),
currentCount+1);
                        break;
```

```
                }
            }
        }

        @Override
        public void shutdownEntity() {

        }

        public int getGatewayDeviceId() {
            return gatewayDeviceId;
        }

        public void setGatewayDeviceId(int gatewayDeviceId) {
            this.gatewayDeviceId = gatewayDeviceId;
        }

        public GeoLocation getGeoLocation() {
            return geoLocation;
        }

        public void setGeoLocation(GeoLocation geoLocation) {
            this.geoLocation = geoLocation;
        }

        public int getUserId() {
            return userId;
        }

        public void setUserId(int userId) {
            this.userId = userId;
        }

        public String getAppId() {
```

```
        return appId;
}

public void setAppId(String appId) {
        this.appId = appId;
}

public String getMyActuatorType() {
        return actuatorType;
}

public void setMyActuatorType(String actuatorType) {
        this.actuatorType = actuatorType;
}

public MyApplication getApp() {
        return app;
}

public void setApp(MyApplication app) {
        this.app = app;
}

public double getLatency() {
        return latency;
}

public void setLatency(double latency) {
        this.latency = latency;
}}
```

MyModulePlacement.java

```
package org.fog.placement;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import org.cloudbus.cloudsim.core.CloudSim;
import org.fog.application.AppModule;
import org.fog.application.MyApplication;
import org.fog.entities.MyActuator;
import org.fog.entities.MyFogDevice;
import org.fog.entities.MySensor;


public class MyModulePlacement extends MyPlacement{

	protected ModuleMapping moduleMapping;
	protected List<MySensor> sensors;
	protected List<MyActuator> actuators;
	protected String moduleToPlace;
	protected Map<Integer, Integer> deviceMipsInfo;



	public MyModulePlacement(List<MyFogDevice> fogDevices, List<MySensor> sensors, List<MyActuator> actuators,
			MyApplication application, ModuleMapping moduleMapping, String moduleToPlace){
		this.setMyFogDevices(fogDevices);
		this.setMyApplication(application);
		this.setModuleMapping(moduleMapping);
		this.setModuleToDeviceMap(new HashMap<String, List<Integer>>());
```

```
            this.setDeviceToModuleMap(new HashMap<Integer,
List<AppModule>>());
            setMySensors(sensors);
            setMyActuators(actuators);
            this.moduleToPlace = moduleToPlace;
            this.deviceMipsInfo = new HashMap<Integer, Integer>();
            mapModules();
        }

        @Override
        protected void mapModules() {

            for(String deviceName :
getModuleMapping().getModuleMapping().keySet()){
                for(String moduleName :
getModuleMapping().getModuleMapping().get(deviceName)){
                    int deviceId = CloudSim.getEntityId(deviceName);
                    AppModule appModule =
getMyApplication().getModuleByName(moduleName);
                    if(!getDeviceToModuleMap().containsKey(deviceId))
                    {
                        List<AppModule>placedModules = new
ArrayList<AppModule>();
                        placedModules.add(appModule);
                        getDeviceToModuleMap().put(deviceId,
placedModules);

                    }
                    else
                    {
                        List<AppModule>placedModules =
getDeviceToModuleMap().get(deviceId);
                        placedModules.add(appModule);
```

```
                                        getDeviceToModuleMap().put(deviceId,
placedModules);
                                }
                        }
                }



                for(MyFogDevice device:getMyFogDevices())
                {
                        int deviceParent = -1;
                        List<Integer>children = new ArrayList<Integer>();

                        if(device.getLevel()==1)
                        {
                                if(!deviceMipsInfo.containsKey(device.getId()))
                                        deviceMipsInfo.put(device.getId(), 0);
                                deviceParent = device.getParentId();
                                for(MyFogDevice deviceChild:getMyFogDevices())
                                {
                                        if(deviceChild.getParentId()==device.getId())
                                        {
                                                children.add(deviceChild.getId());
                                        }
                                }

                                Map<Integer, Double>childDeadline = new
HashMap<Integer, Double>();
                                for(int childId:children)

        childDeadline.put(childId,getMyApplication().getDeadlineInfo().get(childId).
get(moduleToPlace));
```

```
List<Integer> keys = new ArrayList<Integer>(childDeadline.keySet());

for(int i = 0; i<keys.size()-1; i++)
{
    for(int j=0;j<keys.size()-i-1;j++)
    {

if(childDeadline.get(keys.get(j))>childDeadline.get(keys.get(j+1)))
        {
            int tempJ = keys.get(j);
            int tempJn = keys.get(j+1);
            keys.set(j, tempJn);
            keys.set(j+1, tempJ);
        }
    }
}


int baseMipsOfPlacingModule = (int)getMyApplication().getModuleByName(moduleToPlace).getMips();
for(int key:keys)
{
    int currentMips = deviceMipsInfo.get(device.getId());
    AppModule appModule = getMyApplication().getModuleByName(moduleToPlace);
    int additionalMips = getMyApplication().getAdditionalMipsInfo().get(key).get(moduleToPlace);

if(currentMips+baseMipsOfPlacingModule+additionalMips<device.getMips())
    {
```

```
                                        currentMips =
currentMips+baseMipsOfPlacingModule+additionalMips;
                                        deviceMipsInfo.put(device.getId(),
currentMips);

        if(!getDeviceToModuleMap().containsKey(device.getId()))
                                        {

        List<AppModule>placedModules = new ArrayList<AppModule>();
                                                placedModules.add(appModule);

        getDeviceToModuleMap().put(device.getId(), placedModules);

                                        }
                                        else
                                        {

        List<AppModule>placedModules =
getDeviceToModuleMap().get(device.getId());
                                                placedModules.add(appModule);

        getDeviceToModuleMap().put(device.getId(), placedModules);
                                        }
                                }
                                else
                                {
                                        List<AppModule>placedModules =
getDeviceToModuleMap().get(deviceParent);
                                        placedModules.add(appModule);

        getDeviceToModuleMap().put(deviceParent, placedModules);
                                }
                        }
```

```
                    }

                }

        }

        public ModuleMapping getModuleMapping() {
                return moduleMapping;
        }

        public void setModuleMapping(ModuleMapping moduleMapping) {
                this.moduleMapping = moduleMapping;
        }


        public List<MySensor> getMySensors() {
                return sensors;
        }

        public void setMySensors(List<MySensor> sensors) {
                this.sensors = sensors;
        }

        public List<MyActuator> getMyActuators() {
                return actuators;
        }

        public void setMyActuators(List<MyActuator> actuators) {
                this.actuators = actuators;
        }

}
```

MyPlacement.java

```
package org.fog.placement;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import org.cloudbus.cloudsim.core.CloudSim;
import org.fog.application.AppModule;
import org.fog.application.MyApplication;
import org.fog.entities.MyFogDevice;

public abstract class MyPlacement {


	public static int ONLY_CLOUD = 1;
	public static int EDGEWARDS = 2;
	public static int USER_MAPPING = 3;

	private List<MyFogDevice> fogDevices;
	private MyApplication application;
	private Map<String, List<Integer>> moduleToDeviceMap;
	private Map<Integer, List<AppModule>> deviceToModuleMap;
	private Map<Integer, Map<String, Integer>> moduleInstanceCountMap;

	protected abstract void mapModules();

	protected boolean canBeCreated(MyFogDevice fogDevice, AppModule module){
		return fogDevice.getVmAllocationPolicy().allocateHostForVm(module);
	}
```

```
	protected int getParentDevice(int fogDeviceId){
		return
((MyFogDevice)CloudSim.getEntity(fogDeviceId)).getParentId();
	}

	protected MyFogDevice getMyFogDeviceById(int fogDeviceId){
		return (MyFogDevice)CloudSim.getEntity(fogDeviceId);
	}

	protected boolean createModuleInstanceOnDevice(AppModule _module, final
MyFogDevice device, int instanceCount){
		return false;
	}

	protected boolean createModuleInstanceOnDevice(AppModule _module, final
MyFogDevice device){
		AppModule module = null;
		if(getModuleToDeviceMap().containsKey(_module.getName()))
			module = new AppModule(_module);
		else
			module = _module;

		if(canBeCreated(device, module)){
			System.out.println("Creating "+module.getName()+" on device
"+device.getName());

			if(!getDeviceToModuleMap().containsKey(device.getId()))
				getDeviceToModuleMap().put(device.getId(), new
ArrayList<AppModule>());
			getDeviceToModuleMap().get(device.getId()).add(module);


	if(!getModuleToDeviceMap().containsKey(module.getName()))
```

```
                    getModuleToDeviceMap().put(module.getName(), new
ArrayList<Integer>());

        getModuleToDeviceMap().get(module.getName()).add(device.getId());
                    return true;
                } else {
                    System.err.println("Module "+module.getName()+" cannot be
created on device "+device.getName());
                    System.err.println("Terminating");
                    return false;
                }
        }

        protected MyFogDevice getDeviceByName(String deviceName) {
                for(MyFogDevice dev : getMyFogDevices()){
                    if(dev.getName().equals(deviceName))
                        return dev;
                }
                return null;
        }

        protected MyFogDevice getDeviceById(int id){
                for(MyFogDevice dev : getMyFogDevices()){
                    if(dev.getId() == id)
                        return dev;
                }
                return null;
        }

        public List<MyFogDevice> getMyFogDevices() {
                return fogDevices;
        }

        public void setMyFogDevices(List<MyFogDevice> fogDevices) {
```

```
		this.fogDevices = fogDevices;
	}

	public MyApplication getMyApplication() {
		return application;
	}

	public void setMyApplication(MyApplication application) {
		this.application = application;
	}

	public Map<String, List<Integer>> getModuleToDeviceMap() {
		return moduleToDeviceMap;
	}

	public void setModuleToDeviceMap(Map<String, List<Integer>>
moduleToDeviceMap) {
		this.moduleToDeviceMap = moduleToDeviceMap;
	}

	public Map<Integer, List<AppModule>> getDeviceToModuleMap() {
		return deviceToModuleMap;
	}

	public void setDeviceToModuleMap(Map<Integer, List<AppModule>>
deviceToModuleMap) {
		this.deviceToModuleMap = deviceToModuleMap;
	}

	public Map<Integer, Map<String, Integer>> getModuleInstanceCountMap() {
		return moduleInstanceCountMap;
	}
```

```
	public void setModuleInstanceCountMap(Map<Integer, Map<String, Integer>> moduleInstanceCountMap) {
		this.moduleInstanceCountMap = moduleInstanceCountMap;
	}

}
```